Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Rupnik

Interaktivni 3D prikaz terena Slovenije na spletu

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Saša Divjak
Somentor: doc. dr. Matija Marolt

Ljubljana, 2017

Zahvala

Rad bi se zahvalil sošolcem za skupno borbo vse do konca študentskih let, asistentu dr. Cirilu Bohaku za strokovno spremljanje v začetni fazi, somentorju doc. dr. Matiji Maroltu za vodenje k zaključku magistrskega dela in ženi Petri za stalno podporo in motivacijo.

Rok Rupnik, 2017

Contents

Seznam uporabljenih kratic

kratica	angleško	slovensko
ARSO	Slovenian environment agency	Agencija Republike Slovenije za okolje
API	Application Programming Interface	vmesnik za namensko programiranje
DOM	Document Object Model	model objektov dokumenta
GPU	Graphics Processing Unit	grafična procesna enota
GIS	Geographical Information System	geografski informacijski sistem
DMR	Digital Elevation Model	digitalni model reliefa
CPE	Central Processing Unit	centralna procesna enota
GPE	Graphical Processing Unit	grafična procesna enota
RAM	Random Access Memory	spomin z naključnim dostopom (delovni spomin)
OS	Operating System	operacijski sistem
TPS	Thin Plate Spline	zlepki tankih plošč
URL	Uniform Resource Locator	enolični krajevnik vira

Povzetek

Naslov: Interaktivni 3D prikaz terena Slovenije na spletu

V sklopu magistrskega dela smo implementirali interaktivni 3D prikaz digitalnega modela terena Slovenije na spletu. Model je izdelala Agencija Republike Slovenije za okolje s pomočjo tehnologije LIDAR. Velikost vseh podatkov o višini v skrčeni obliki znaša približno 60 GB. Take količine podatkov ne moremo neposredno prenašati za prikaz v brskalniku, zato smo jih morali poenostaviti in organizirati v več nivojev podrobnosti. Implementirali smo strežniški program, ki podatke na zahtevo streže aplikaciji v brskalniku odjemalca, ta pa skrbi za njihovo upodabljanje. Po implementaciji upodabljanja smo primerjali rešitev, ki uporablja obstoječe knjižnice, in lastno tehniko, ki je sledila metodi CDLOD.

Ključne besede

3D vizualizacija terena, spletne tehnologije, WebGL, LOD, CDLOD, LIDAR

Abstract

Title: Interactive online 3D display of Slovenian terrain

In this master thesis we implemented an online interactive 3D display of a digital terrain model of Slovenia. The model was made with the help of LIDAR technology by Slovenian environment agency (ARSO). The size of all data in compressed form is about 60 GB. We cannot transfer this amount of data directly for rendering in the browser. We simplified the data and organised it into multiple levels of detail. This data was then served from a server program to an app in the client's browser. This app made sure to load and render the correct data. After the implementation we compared the solution from existing libraries and our own technique which was based on the CDLOD method.

Keywords

3D terrain visualisation, web tehnologies, WebGL, LOD, CDLOD, LIDAR

Poglavje 1

Uvod

1.1 Predstavitev problema

V računalništvu so 3D vizualizacije že dalj časa močno prisotne. Uporabljamo jih tako v komercialne namene (računalniške igre) kot tudi v znanosti. 3D vizualizacije nam koristijo povsod, kjer se obdeluje prostorske podatke, kot na primer v strojništvu, fiziki in zdravstvu. Uporabo tehnologij za upodabljanje 3D vizualizacij nam olajšuje razvoj grafičnih procesnih enot (GPE), saj lahko že v povprečnem prenosniku najdemo zmogljive grafične procesne enote.

Pogoste so tudi 3D vizualizacije terena, sploh v računalniških igrah, kjer predstavljajo osnovo za prikazovanje prostora okrog igralca. V igrah in podobnih programih je upodabljanje olajšano zaradi lažjega dostopa do podatkov, ki se lahko nahajajo na istem disku, ter zaradi večjih zmožnosti uporabe GPE, do katere imamo bolj neposreden dostop kot v programih, ki jih izvajamo v brskalniku. Nekoliko večji izziv predstavlja upodabljanje terena iz narave, sploh, ko se to upodabljanje odvija na spletu. Nekatere obstoječe implementacije prikazovanja terena uspešno rešujejo te izzive, čeprav s svojimi rešitvami vpeljejo nekaj omejitev; več o tem v Pregledu področja.

Motivacija tega magistrskega dela je iskanje učinkovite rešitve za implementacijo interaktivnega prikazovanja realnega terena Slovenije v 3D razsežnosti na spletu. Težavnost zadanega izziva izhaja iz dejstva, da gre za veliko količino podatkov (60 GB v skrčeni obliki), zato bo med obstoječimi rešitvami potrebno poiskati primerne in jih ustrezno prilagoditi. Izbira območja za prikaz je temeljila na dejstvu, da je za teren Slovenije dostopen natančen digitalni model reliefa, ki ga je izdelala Agencija Republike Slovenije za okolje s pomočjo tehnologije LIDAR.

Glavni cilj magistrskega dela je najprej interaktivno upodabljati veliko količino podatkov LIDAR. Podatke bomo upodabljali s poligonsko mrežo, na kateri bomo prikazali teksturo z ortofoto posnetki. Za doseganje tega cilja bo najprej pomembno, da čim bolj zmanjšamo čas nalaganja podatkov, pri čemer nas ovira visoka natančnost modela reliefa, ki ga želimo prikazati. Za prikaz celotnega modela bo torej potrebno nekoliko poenostaviti podatke. Kasneje želimo doseči čim večjo učinkovitost upodabljanja in obenem teren prikazati kar se da natančno - izziv bo v usklajevanju teh dveh ciljev. Zadnji cilj je razviti primerno implementacijo upodabljanja nivojev podrobnosti (LOD), ki bo omogočala zadovoljivo interaktivnost aplikacije. V naslednjem delu tega poglavja bomo predstavili nekaj metod, ki se spopadajo s temi izzivi.

1.2 Pregled področja

Na razširjenost upodabljanja 3D objektov v računalništvu kaže veliko število člankov, ki se podrobneje ukvarjajo s to tematiko. V pregledu področja bomo obravnavali tiste, ki se približajo našemu problemu in obdelajo poenostavljanje strukture 3D podatkov, upodabljanje 3D grafike v brskalniku in upodabljanje nivojev podrobnosti.

1.2.1 Poenostavljanje podatkov

Avtorji članka [1] predstavijo in kvantitativno primerjajo več metod za poenostavljanje 3D geometrij. Analizo so izvedli na svoji implementaciji inkrementalnega in hierarhičnega gručenja, iterativnega poenostavljanja ter na implementaciji algoritma simulacije delcev. S temi metodami so ustvarili približke originalnih modelov. Vse omenjene metode delujejo direktno na oblaku točk in ne zahtevajo vmesne mozaične razporeditve (angl. tesselation). Avtorji pokažejo, kako se lahko uporabi lokalno oceno variance in druge metrike za zmanjšanje napake približka ter za koncentriranje večih točk v regijah z večjo ukrivljenostjo. Sami so tudi implementirali novo metodo za računanje numeričnih in vizualnih napak pridobljenih približkov. Podobno v članku [2] avtorji pregledajo in opišejo več algoritmov za poenostavljanje majhnih, oddaljenih ali drugače nepomembnih delov poligonskih geometrij. Iz obeh člankov ugotavljamo, da so omenjene metode uspešne in da dosegajo odlične rezultate, vendar bi implementacija teh metod za kodiranje podatkov za sabo potegnila veliko zahtevnejšo implementacijo dekodiranja podatkov v brskalniku oz. na GPE. Iz navedenih razlogov smo se odločili za preprostejše kodiranje podatkov, ki je opisano v nadaljnjih poglavjih.

1.2.2 Upodabljanje prostorskih podatkov

Primer interaktivnega upodabljanja 3D podatkov v realnem času na napravi spletnega odjemalca je prikazan v članku [3]. Avtorji poudarjajo, da njihov sistem z uporabo procesorske moči odjemalca pridobi na skupni sposobnosti upodabljanja in poleg tega eliminira škodljiv učinek nekaterih točk ozkega grla, kot je na primer omrežje. Sistem s pridom izkorišča zmožnosti standarda WebGL, ki razvijalcu aplikacije za brskalnik olajša dostop do GPE. S tem se odstrani potrebo po posebej prirejeni strojni opremi, sistem pa je kljub temu sposoben dosegati učinkovit nivo upodabljanja. Avtorji ugotavljajo, da splet čedalje bolj intenzivno tekmuje z namiznimi aplikacijami na številnih

področjih, ampak v računalniški grafiki so spletni razvijalci omejeni z zahtevnostjo nekaterih aplikacij, kot so interaktivne vizualizacije prostorninskih podatkov v znanosti. V svojem članku so obravnavali učinkovitost in skalabilnost upodabljanja prostorninskih podatkov s tehniko metanja žarkov. Omenjena metoda za naš primer ni bila primerna, ker bi prinesla preveč računsko zahtevnih operacij, ki bi se ponavljale ob vsakem premiku. Prikazovali so podatke iz medicine in radarske meteorologije, medtem ko so avtorji članka [4] razvili samostojno knjižnico za prikaz zahtevnih molekularnih struktur v brskalniku. Odločili so se za neposredno uporabo standarda WebGL, brez uporabe morebitnih knjižnic. Tako so lahko povečali učinkovitost prikazanih vizualizacij in obenem izpostavili lasten vmesnik uporabniškega programa (angl. API). Metoda iz članka [5] uporablja celo tehnologijo spletnih vtičnic (angl. Web Sockets) za optimizacijo pošiljanja podatkov v aplikacijo.

V članku [6] so avtorji leta 1998 ugotavljali, da je breme upodabljanja trianguliranih površin v realnem času na grafično procesni enoti potrebno nadzorovati z adaptivno triangulacijo površja in z izkoriščanjem različnih nivojev natančnosti. Avtorji predlagajo enoten sistem za vizualizacijo, ki združuje adaptivno triangulacijo, dinamično upravljanje scene in urejanje prostorskih podatkov. Model triangulacije je osnovan na omejeni triangulaciji štiriškega drevesa, za katerega predlagajo tudi nove algoritme triangulacije. Ti vključujejo oceno napake, delo s progresivnimi mrežami in izboljšave v učinkovitosti in prostorski dostop.

Leta 2000 so avtorji članka [7] predstavili novo tehniko upodabljanja in procesiranja več tekstur v različnih ločljivostih pri prikazovanju modelov terena z različnimi nivoji natančnosti. Opisujejo apliciranje te tehnike na zasnovi interaktivnega animiranega terena. Pristop je osnovan na modelu za teksturo terena, ki se dopolnjuje z modelom za geometrijo terena. Oba modela sta podana v več resolucijah. Za vsak različen sloj se zgradi piramida slik in drevo tekstur. Več slojev teksture je lahko povezanih z enim modelom

terena in lahko se jih združuje na različne načine. Algoritem upodabljanja simultano pregleda modela tekstur in geometrije ter upošteva geometrične napake in napake aproksimacije tekstur. Algoritem uporablja upodabljanje v več prehodih in izkorišča uporabo več tekstur za delovanje v realnem času. Aplikacije algoritma vključujejo interaktivne animacije tekstur in topografske teksture. Te tehnike so v tistem času pomenile ogromen potencial za razvoj aplikacij za predstavitev, raziskovanje in upravljanje s prostorsko-časovnimi podatki.

Kmalu zatem so se že pojavili prvi poskusi vizualizacije terena preko spleta. Avtorji članka [8] ugotavljajo, da so pogoji za vizualizacijo terena v 3D preko spleta katastrofalni za uporabnike s šibko pasovno širino zaradi velike količine podatkov, ki jih te vizualizacije vključujejo. V tem članku predlagajo in implementirajo metodo, ki temelji na izbiranju razrezanih delov terena. Poleg tega ta metoda omogoča prilagojene poizvedbe v svojem geografskem informacijskem sistemu. Zasnovali in razvili so tudi prototip spletnega geografskega informacijskega sistema, ki ponuja hibridni uporabniški vmesnik z 2D in 3D zmožnostmi prikazovanja.

V bolj sodobnem članku [9] iz leta 2014 avtorji predlagajo nov sistem za shranjevanje podatkov o terenu. Njihova motivacija je bila dejstvo, da kljub razširjenosti spletnih aplikacij, ki upodabljajo teren, zaradi povečanega števila tehnologij zaznavanja narašča tudi količina podatkov, ki jo je potrebno prenašati v spletnih vizualizacijah, če jih želimo izvajati na interaktivnem nivoju. Avtorji predlagajo nov sistem shranjevanja podatkov, kjer teren razdelijo na kvadrate, znotraj katerih v posamezni točki beležijo zgolj odstopanje od neke povprečne vrednosti. S tem načinom shranjevanja bi zamenjali tradicionalno piramido terena v večih resolucijah. V primerjavi s trenutno aktualnimi pristopi njihova metoda dosega višjo stopnjo kompresije podatkov o terenu, s čimer zmanjšajo prostor za shrambo in pasovno širino zahtevano za prenos podatkov. Poleg tega dosegajo boljšo vizualno kvaliteto

virtualnega terena, ko se rekonstruira po dekompresiji podatkov.

1.2.3 Upodabljanje podatkov LIDAR

Podatki, zajeti s tehnologijo LIDAR, se običajno shranjujejo v oblaku točk. Iz tega razloga metode za njihovo upodabljanje izbirajo upodabljanje LIDAR podatkov z različno velikimi točkami v prostoru. Tej tehniki sledi implementacija opisana v članku [10], kjer avtorji predpišejo sistem za progresivno upodabljanje mrež z več sto milijoni poligonov. Njihov sistem temelji na omejitvenih kroglah in upodabljanju s točkami. Uporabljajo zgolj eno podatkovno strukturo za izločanje točk izven vidnega polja, preverjanje orientacije poligonov, izbor točk glede na nivo podrobnosti in končno upodabljanje. Predstavitev objektov je kompaktna in se jo lahko hitro izračuna, kar je uporabno na večjih zbirkah podatkov. Njihova implementacija, spisana za 3D projekte velikih razsežnosti, se hitro zažene in vzpostavi interaktiven prikaz z visoko stopnjo sličic na sekundo, neodvisen od kompleksnosti objektov in pozicije kamere. Sistem zagotavlja spodobno kvaliteto slike med gibanjem in se med mirovanjem posodablja do končne kvalitete slike. Sistem so demonstrirali na objektih, ki so vsebovali stotine milijonov elementov.

Opis bolj geografsko usmerjenega projekta najdemo v članku [11], v katerem so obdelani, analizirani in nazadnje prikazani lasersko skenirani in fotogrametrijsko pridobljeni podatki. Svojo zasnovo prikažejo na treh okoljskih aplikacijah: ocenjevanje škode gozdnih neviht, ocenjevanje prostorninskih sprememb v odprtem rudniku in 3D vizualizacija modela mesta. Eden od glavnih ciljev članka je bilo preučiti uporabnost tako obsežnih podatkov, ki so vsem dostopni. Ugotavljajo, da je veliko možnosti za prikazovanje teh podatkov, vendar vidijo oviro v pomanjkanju standardizacije pridobljenih podatkov, ki jim težko pripišemo oceno o njihovi kvaliteti. Nekaj drugačnih načinov vizualizacije podatkov LIDAR je opisanih v članku [12]. S temi tehnikami avtorji odkrivajo arheološko pomembne značilnosti v visoko ločljivostnih di-

gitalnih modelih reliefa, ki so izdelani na podlagi podatkov LIDAR. Ugotavljajo, da so za različne vrste terena primerne različne tehnike prikazovanja.

1.2.4 Nivoji natančnosti

Nekoliko bližje ciljem našega magistrskega dela pridejo metode opisane v člankih [13] in [14], kjer avtorji upodabljajo geografske podatke na terenu, v drugem članku celo upoštevajo časovno komponento podatkov, saj so implementirali 4D prikaz pomorskih prostorskih podatkov. Večjih geografskih območij so se uspešno lotili avtorji članka [15], ki vpeljejo naprednejše tehnike skrčevanja podatkov in grajenje mreže za prikaz več nivojev podrobnosti. Količino podatkov v štiriškem drevesu so zmanjšali s shranjevanjem samo novo dodanih informacij na posameznem nivoju. Svojo metodo so implementirali za namizno aplikacijo in posledično niso imeli težav s prenašanjem podatkov preko medmrežja.

Avtorji članka [16] obravnavajo lokalno prilagajanje geometrične kompleksnosti spreminjajočim parametrom pogleda kot ključni vidik upodabljanja velikih površin v realnem času. V času izida članka je bilo razvitih že več tehnik za reševanje problema upodabljanja v več nivojih natančnosti v odvisnosti od pogleda. Med temi izpostavljajo ogrodje progresivne mreže, ki je odvisna od pogleda (angl. View-dependent Progressive Mesh - VDPM), ki predstavlja poljubno mrežo trikotnikov kot hierarhijo geometrično optimiziranih transformacij za izboljšanje modela. Iz take mreže lahko učinkovito pridobimo mreže različnih kakovosti. V tem članku avtorji predlagajo številne izboljšave mreže VDPM, s katerimi odstranijo štrleče artefakte in tako zgladijo prikazano geometrijo. Izhod postopka, ki so ga razvili, prinaša s seboj tudi zmanjšano velikost shranjenih podatkov.

V članku [17] iz leta 2015 avtorji predstavijo nov kriterij za nivo natančnosti na osnovi tradicionalnega algoritma, ki na terenu zgradi štiriško

drevo. Pri upodabljanju terena na mobilnih napravah so pozorni na pomembnejše kriterije in v svoji metodi poskrbijo, da se algoritem na vsaki napravi izvaja v skladu z njenimi omejitvami, še posebej se ozira na porabo spomina in zmogljivost grafične strojne opreme. Istočasno njihov sistem brez težav poskrbi za tipično interakcijo na zaslonu s strani uporabnika. Na koncu predstavijo uporabno študijo učinkovitosti, kjer primerjajo nekaj mobilnih in namiznih naprav ter se osredotočajo na tehnike nivojev natančnosti, testov vidnosti, delitev tekstur, nalaganju deljenih tekstur na grafično procesno enoto in samo upodabljanje.

Podobno tehniko vpelje članek [18], ki se je tudi najbolj približal ciljem naše implementacije, saj so bili elementi tehnike iz članka, za razliko od sorodnih metod, najbolj primerni za implementacijo v spletnem okolju. Članek predstavlja tehniko upodabljanja terena z višinsko mapo (angl. heightmap), ki nadgradi nekaj obstoječih metod z novimi idejami. Namesto enakomerno razdeljene mreže uporablja prilagojeno različico mreže štiriškega drevesa, ki se premika po terenu in tako zagotavlja konstantno kvaliteto prikaza opazovalcu. Glavna izboljšava metode je v tem, da je funkcija nivojev natančnosti ista čez celotno prikazano mrežo in je osnovana v natančni razliki med terenom in opazovalcem. Za doseganje teh zahtev so uvedli novo tehniko za prehajanje med posameznimi nivoji natančnosti. Z uvedbo te tehnike je sistem bolj predvidljiv in zanesljiv, ima boljšo razporeditev trikotnikov po zaslonu, bolj čiste prehode med nivoji natančnosti in posledično nima potreb po šivanju mreže. Te izboljšave tudi poenostavljajo integracijo z drugimi sistemi za upravljanje nivojev natančnosti. Tudi iz vidika učinkovitosti je ta pristop ugodnejši od primerljivih tehnik na grafično procesni enoti. Pristop deluje na grafični strojni opremi, ki podpira senčilnike tipa Shader Model 3.0 in višje.

1.3 Namen naloge

Izmed obravnavanih člankov je na naše kriterije najbolje odgovarjal članek [18] in njegovo metodo smo delno implementirali v naši rešitvi. Iz tega članka smo namreč lahko implementirali tehniko grajenja mreže ploščic, na katerih smo kasneje upodabljali geometrijo na več nivojih podrobnosti. Ta tehnika grajenja ploščic je prinesla dovolj prednosti pristopa z več nivoji podrobnosti, ampak obenem ni zahtevala težavnega dekodiranja ali podobnih zahtevnih operacij na CPE in GPE. Več o uporabljenih pristopih iz te metode sledi v poglavju o teoretičnem izhodišču.

Doprinos magistrskega dela bo v tem, da bomo implementirali metodo upodabljanja terena z več nivoji podrobnosti, ki sledi metodi v članku [18]. Metodo bomo prilagodili za upodabljanje realnega terena na podlagi zelo natančnih podatkov in na koncu izmerili njeno učinkovitost. Merjenje učinkovitosti bo izvedeno s primerjanjem naše prilagojene metode upodabljanja na več nivojih natančnosti z metodo, ki uporablja že pripravljene objekte, namenjene prikazovanju geometrij na več nivojih natančnosti. Razlika med metodama je predvsem v tem, da se naša prilagojena metoda v svojem jedru izvaja na grafično procesni enoti (GPE), medtem ko se dober del preračunavanja metode, ki temelji na pripravljenih objektih iz knjižnice, izvaja na centralno procesni enoti (CPE).

Osredotočili se bomo na kriterije omenjene v ciljih magistrske naloge. Predvidevamo začetni čas nalaganja aplikacije reda velikosti 10 s in omejitev v začetku naloženih podatkov na nekaj MB. Za zadovoljivo upodabljanje bomo pričakovali prikazovanje terena z vsaj 20 sličicami na sekundo, tudi ob povečanem pogledu tik ob modelu reliefa ter celo ob pogledu proti obzorju, kar predstavlja največji preizkus za prikaz večih nivojev podrobnosti, s katerim želimo doseči visoko stopnjo interaktivnosti aplikacije.

1.4 Struktura magistrskega dela

V naslednjem poglavju je v teoriji razloženih nekaj osnovnih pojmov, pristopov in tehnik, na katerih temelji to magistrsko delo. Omenjeni so tudi nekateri geodetski pojmi, ki jih potrebujemo kasneje v implementaciji.

V poglavju 3 natančneje pojasnimo načrtovanje in potek implementacije naše rešitve. V tem poglavju bo govora o upravljanju s podatki, prikazovanju terena v brskalniku in o posodabljanju pogleda, ki se prikazuje v uporabniškem vmesniku.

Na koncu sledijo v poglavju 4 analize implementirane rešitve, ki jih ovrednotimo v poglavju 5, kjer bomo podali sklepne ugotovitve.

Poglavje 2

Teoretično izhodišče

Na tem mestu bomo zapisali teoretično ozadje konceptov in tehnologij, ki smo jih uporabljali. Na začetku se bomo dotaknili bolj geodetskih pojmov, ki razlagajo, kako je bil izdelan naš model reliefa, ki ga prikazujemo. Kasneje bomo opisali stiskanje podatkov, s katerimi smo upravljali. Na koncu bomo predstavili še ozadje upodabljanja terena in upravljanja z nivoji podrobnosti.

2.1 LIDAR

Tehnologija LIDAR (angl. light detection and ranging) se danes pogosto uporablja za meritve v arheologiji, geografiji, geologiji, geomorfologiji, seizmologiji in drugih vedah, ki potrebujejo izvajanje meritev na širših geografskih območjih. To je tehnologija optičnega daljinskega zaznavanja, ki za delovanje uporablja lasersko svetlobo, ki jo iz letala med preletom usmerimo proti zemlji. Na podlagi razlike v času od začetka svetlobnega signala do vrnitve odbitega signala se lahko izračuna razdalja in drugi parametri tarče, ki smo jo zadeli s svetlobnim signalom. Druga imena za LIDAR vključujejo ALSM (angl. Airborne Laser Swath Mapping) in lasersko altimetrijo.

Delovanje tehnologije spominja na radar, razlika je na primer v tem, da

se pri tehnologiji LIDAR uporabljajo mnogo krajše valovne dolžine elektromagnetnega spektra. V splošnem velja, da lahko pri merjenju z elektromagnetnim sevanjem zaznamo predmete v velikosti valovne dolžine ali večje, saj mora predmet predstavljati dovolj veliko oviro v signalu, da se ustvari odboj. Pri radarskih frekvencah npr. kovinski predmeti povzročajo precejšnje odboje, medtem ko nekovinski, sploh pa manjši predmeti, sploh ne povzročijo odboja ali pa je le-ta šibek. Ker je valovna dolžina pri laserju toliko manjša, lahko s tehnologijo LIDAR izvajamo toliko bolj natančne meritve. Laser ima tudi toliko bolj ozek žarek, kar omogoča izdelavo meritev z veliko večjo resolucijo kot pri radarju. Poleg tega veliko snovi močneje reagira na elektromagnetno valovanje z valovno dolžino v vidnem polju, kot na svetlobo mikro valov, ki jo oddaja radar. To se odraža v lažjem zaznavanju teh materialov [19].

Primer uporabe tehnologije LIDAR lahko najdemo v laserskem skeniranju gozdov, s katerim se pridobi natančen model terena. Obenem se lahko oceni več podatkov o količini različnih gozdnih virov. V času pisanja članka [20] je bilo veliko odprav z laserskim skeniranjem namenjenih zgolj modeliranju terena in ne pridobivanju podatkov povezanih z vegetacijo, kar nakazuje takratno potrebo po standardih in specifikacijah za pridobivanje drugih meritev iz podatkov LIDAR. Avtorji članka predstavijo pet preprostih in lahko razumljivih modelov, pridobljenih s tehnologijo LIDAR, ki so povezani z gozdarstvom. Z njimi želijo zagotoviti, da se upošteva potrebe gozdarstva pri načrtovanju prihodnjih letalskih odprav z laserskim skeniranjem.

Metodam izdelave digitalnega modela reliefa in odstranjevanja rastja ter drugih elementov, ki se pojavijo v meritvah s tehnologijo LIDAR, se je posvečal že leta 2000 objavljeni članek [21]. Avtorji ugotavljajo, da so že v tistem času LIDAR podatki omogočali izdelavo mnogo bolj natančnih digitalnih modelov reliefa v primerjavi z drugimi tehnologijami.

Kljub visoki ceni izdelave digitalnih modelov reliefa so se taki izdelki v času članka [22] pogosto uporabljali brez zadovoljive ocene natančnosti podanih podatkov. V večini primerov je ocena podana s primerjanjem LIDAR višin s končnim neodvisnim vzorcem točk višje natančnosti, pri čemer se predpostavlja normalno porazdelitev napake. V članku [22] avtorji predlagajo nove metode za ocenjevanje vertikalne natančnosti digitalnih modelov reliefa pridobljenih s tehnologijo LIDAR.

2.1.1 Meritve LIDAR s strani ARSO

Taka in podobna prizadevanja ([23][24]) za definiranje standardov in postopkov so pripravila temelje za izvajanje natančnejših meritev v času, ko je meritve izvajala Agencija Republike Slovenije za okolje [25]. Meritve so bile načrtovane za leti 2011 in 2012. Izvajane so bile v več posameznih blokih, ki so skupaj pokrili območje Slovenije. Nekateri bloki so bili merjeni v višji ločljivosti (10 točk na m^2), ti so pokrivali plazovita in poplavna območja. Območja večjih gozdov in gora so bila merjena v najnižji ločljivosti (2 točki na m^2), medtem, ko je bila večina Slovenije merjena v vmesni ločljivosti (5 točk na m^2). Pri načrtovanju terminov letalskih odprav so se upoštevale tudi naravne zakonitosti, saj je bila večina meritev izvedena v pomladnem času, ko je zalistanost terena nizka.

Eden od rezultatov meritev je najnatančnejši digitalni model reliefa Slovenije doslej. Pri načrtovanju modela se je zahtevalo natančnost 30 cm v vodoravni smeri in 15 cm v navpični smeri. Načrtovano je podajanje točk na mreži 1 x 1 m. Prejšnji model reliefa je imel točke podane na mreži 5 x 5 m in zahtevana natančnost je bila 1 m. Pri modelu, ki ga je izdelal ARSO, so koordinate točk podane v novem državnem sistemu D96/TM in so kasneje transformirane v starejši koordinatni sistem Gauss-Krueger (D48) [25].

Poenostavljanje podatkov LIDAR v oblaku točk je opisano v tehničnem

poročilu [26], kjer se pri opisu izdelave digitalnega modela reliefa sklicujejo na metodo opisano v članku [27]. Opisano tehniko so uporabljali na ARSO za izdelavo digitalnega modela reliefa, ki ga upodabljamo. Zajete točke najprej razdelijo v enako velike celice, organizirane v regularno mrežo, ki pokrije celotno območje snemanja. Velikost celice se prilagaja gostoti oblaka točk, ki ga prejmejo kot vhod, kontrolna točka in višina celice je določena s tisto točko, ki najverjetneje opisuje teren, to je z najnižjo točko v njej. Kasneje se v zanki ponavljajo trije koraki, dokler ne dosežemo željene ločljivosti digitalnega modela reliefa. Najprej iz pripravljenih celic zgradimo piramidno strukturo od spodaj navzgor, pri čemer na vsakem koraku združujemo po 4 celice v eno. Nadalje na celicah vrhnjega nivoja pridobljene piramidne strukture z interpolacijo zlepkov tankih plošč (angl. thin plate spline, TPS) zgradijo DMR, ki se mu postopoma viša ločljivost na vsakem izmed nižjih nivojev piramidne strukture celic. Na koncu vsake iteracije se izvede filtriranje točk v modelu preko primerjanja točk iz trenutnega približka DMR in točk iz nižjega nivoja hierarhije celic.

Filtriranje se izvede tako, da primerjamo razlike kontrolnih točk nižjega nivoja hierarhije celic s točkami iz trenutnega modela DMR. Vhodni oblak točk podvzorčimo v mrežo in pregledamo okolico w vsake točke. Vsaki točki poiščemo najnižjo in najvišjo točko iz njene okolice in ji odštejemo razliko med njima. Izračun za točko z indeksom i in j je prikazan v formuli 2.1.

$$\begin{aligned} p_{i,j} &= p_{i,j} - (max(p_{k,l}) - min(p_{k,l})) \\ (i - \frac{w}{2}) &\leq k \leq (i + \frac{w}{2}) \\ (j - \frac{w}{2}) &\leq l \leq (j + \frac{w}{2}) \end{aligned} \tag{2.1}$$

2.2 Poenostavljanje podatkov

Pomemben del implementacije magistrske naloge je bil v obdelavi in shranjevanju podatkov. Pregled tehnik shranjevanja in optimiziranja podatkov povzemamo po članku [28].

Pri večini tehnik za 3D grafiko v brskalniku so podatki shranjeni v poligonskih (običajno trikotnih) mrežah, sestavljenih iz vozlišč in ploskev. Taki podatki so lahko brez optimizacij zelo veliki, sploh kadar opisujejo podrobnosti posameznih objektov. Na primer, izhod laserskega skeniranja velikih 3D objektov lahko obsega več 100 megabajtov. Velike datoteke imajo velik vpliv na učinkovitost sistema, tako iz vidika obdelave, kot tudi iz vidika upodabljanja grafike. Posledično se je pojavilo veliko prizadevanj za optimizacijo in delitev mrež podatkov, ampak na splošno velja, da je večina teh postopkov usmerjenih proti upravljanju specifičnih vrst podatkov in da taki postopki v glavnem niso namenjeni poljubnim mrežam s podatki o normalah in teksturnih koordinatah, t.j. mrežah namenjenih spletnim tehnologijam.

Primer uveljavljenega pristopa h kompresiji in optimizaciji podatkov so progresivne mreže (angl. progressive meshes). Originalno jih je leta 1996 predlagal Hoppe [29]. Dovoljujejo neprekinjeno progresivno izboljševanje poligonske mreže med prenosom podatkov preko interneta. Progresivne mreže delujejo tako, da odjemalec najprej iz strežnika prenese grobo mrežo in nato to mrežo progresivno izboljšuje z operacijami delitve vozlišč, ki jih pretočno prenaša iz strežnika, dokler v celoti ne poustvari mreže v visoki ločljivosti. Številni avtorji so v svojih člankih obravnavali progresivno nalaganje mreže podatkov ([30], [31], [32]) in po analizi njihovih zaključkov lahko izpeljemo nekaj posebnih okoliščin za kompresijo mreže namenjene spletnemu upodabljanju 3D vsebine.

- **Uspešen kompromis med hitrostjo prenosa in zahtevnostjo de-**

kompresije. Tehnike kompresiranja mrež za spletne 3D aplikacije so posebne v tem, da pri učinkovitosti igrata veliko vlogo tako hitrost prenosa kot tudi hitrost dekompresiranja podatkov. Algoritem, ki doseže visoko stopnjo kompresije, ni nujno primeren za spletni sistem, ker se lahko zgodi, da so hitrosti dekompresije nizke. Ta pojav je še posebej relevanten trenutno, ko se, kljub izboljšanim hitrostim prenosa podatkov po medmrežju, vse bolj razširjajo naprave z manjšo zmogljivostjo procesiranja, ker zasledujejo manjšo porabo električne energije.

- **Dekompresija v brskalniku.** Po drugi strani je pri dekompresiji treba upoštevati dejstvo, da se mora pri aplikaciji brez dodatnih vtičnikov dekompresija implementirati v jeziku JavaScript, ki je kljub številnim izboljšavam lahko počasnejši od kode v katerem je bila implementirana kompresija. Sploh, če v algoritmu ne uspemo najti načina, kako dekompresijo implementirati na grafično procesni enoti, ne smemo zanemariti časov izvajanja dekompresije v jeziku JavaScript.

- **Več objektov na sceni.** Tipična 3D scena v spletnem upodabljanju lahko vključuje več objektov z različnimi zahtevami in velikostmi. Lahko vključuje tudi spremembe in animacije teh objektov. Klasične metode progresivnih mrež lahko dobro delujejo s pravilno vzorčenimi in zaprtimi površinami, medtem ko ne delujejo pravilno ali učinkovito v večini ostalih primerov uporabe v spletnih 3D aplikacijah. Če ne drugega, se lahko zgodi, da je potrebno nekaj predprocesiranja za delitev in klasifikacijo mreže pred kompresijo.

- **Različni tipi podatkov za posamezen objekt.** Osnovne progresivne mreže v svojih postopkih ne upoštevajo drugih vrst podatkov kot le lokacijo vozlišč mreže. Čeprav je to najbolj pomembna vrsta informacije, je lahko v mreži shranjenih veliko drugih komponent, ki so pomebne za prikaz, kot so na primer vektorji normal, teksturne koordinate in vrednosti barve.

V letih pred izidom članka [28] se je pojavilo več prizadevanj k naslavljanju teh štirih posebnosti in njihov cilj je bil najti uporabno metodo za kompresiranje mrež, ki bo bolj prilagojena spletnemu prikazu 3D vsebine.

Naša metoda kodiranja je v svoji osnovi najbolj podobna eni od metod, opisanih v članku [1], in sicer iterativnemu poenostavljanju. Ta metoda, kot opisana v članku, iterativno zmanjšuje število točk po predpisanem dogovoru in je zelo podobna metodam za poenostavljanje v progresivnih mrežah. V članku se metoda uporablja z različnimi atomskimi operacijami redčenja točk, ki so organizirane v vrsto glede na metriko napake, ki ovrednoti napako, ki bi jo povzročila izvedba te operacije. Iteracija je nato izvedena tako, da se najprej izvedejo operacije, ki povzročijo najmanjšo napako. Operacije kot argument sprejemajo pare oglišč v_1 in v_2, ki jih zamenjajo z novo točko $\overline{v}$. Napako operacije se oceni na podlagi dejstva, da je v originalnem modelu vsako oglišče presek ravnin, natančneje ravnin, ki jih določajo trikotniki ob tem oglišču in jih razpenjajo vektorji $\mathbf{e}_i = \mathbf{v} - \mathbf{v}_i$ in $\mathbf{b}_i = \mathbf{e}_i \times \mathbf{n}$, pri čemer je $\mathbf{n}$ ocenjeni vektor normale v oglišču $\mathbf{v}$. S tako določeno množico ravnin lahko definiramo napako oglišča v kot vsoto kvadratov razdalj do svojih ravnin, kot je prikazano v enačbi 2.2, pri čemer $\mathbf{p} = [a\ b\ c\ d]^\top$ predstavlja ravnino definirano z enačbo $ax + by + cz + d = 0$, kjer velja $a^2 + b^2 + c^2 = 1$.

$$\Delta(\mathbf{v}) = \Delta([v_x\ v_y\ v_z\ 1]^\top) = \sum_{p\ \in\ ravnine(v)} (\mathbf{p}^\top \mathbf{v})^2 \tag{2.2}$$

Formulo 2.2 lahko prepišemo v kvadratični obliki

$$\begin{aligned} \Delta(\mathbf{v}) &= \sum_{p\ \in\ ravnine(v)} (\mathbf{v}^\top \mathbf{p})(\mathbf{p}^\top \mathbf{v}) \\ &= \sum_{p\ \in\ ravnine(v)} \mathbf{v}^\top (\mathbf{p}\mathbf{v}^\top)\mathbf{v} \\ &= \mathbf{v}^\top \Big(\sum_{p\ \in\ ravnine(v)} \mathbf{K_p} \Big) \mathbf{v}, \end{aligned} \tag{2.3}$$

kjer je $\mathbf{K_p}$ matrika

$$\mathbf{K_p} = \mathbf{p}\mathbf{p}^\top = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}.$$

Z vsoto takih matrik po vseh ravninah lahko predstavimo celo množico ravnin z eno matriko $\mathbf{Q}$. Posledično lahko za oceno napake novega oglišča $\overline{\mathbf{v}}$ namesto združevanja množic ravnin ob starih ogliščih (ravnine $(\mathbf{v}_1) \cup$ ravnine $(\mathbf{v}_2)$) preprosto seštejemo matriki $\mathbf{Q}_1$ in $\mathbf{Q}_2$. Na koncu lahko zapišemo napako oz. ceno novega oglišča $\overline{\mathbf{v}}$ v formuli 2.4. Podrobnejšo analizo mere napake smo našli v članku [33].

$$\Delta(\overline{\mathbf{v}}) = \overline{\mathbf{v}}^\top(\mathbf{Q}_1 + \mathbf{Q}_2)\overline{\mathbf{v}} \tag{2.4}$$

Ker lahko v metodi iterativnega poenostavljanja iz članka [1] zaporedne operacije poenostavljanja izvajamo na različnih lokacijah v mreži, s to metodo ne pridobimo regularne mreže točk. Taka rešitev bi v našem primeru zahtevala manj učinkovito shranjevanje podatkov, saj bi morali za vsako točko hraniti tudi koordinate točk in ne le njihove nadmorske višine. Naša končna rešitev, ki je opisana v kasnejših odstavkih, temelji na bolj enostavnem algoritmu poenostavljanja mreže, ki ohranja regularno mrežo točk. Bolj splošno poenostavljanje nadgradimo z uporabo učinkovitega kodiranja podatkov v png slike, s katerimi je kasneje tudi lažje delati v naši strukturi sistema za nalaganje podatkov v več nivojih natančnosti.

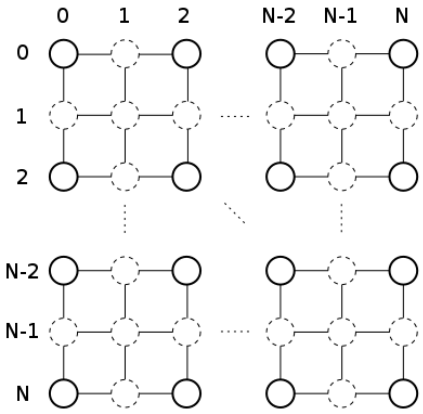
Za pripravo podatkov o terenu na nalaganje v več nivojih natančnosti smo si zamislili strukturo zakodiranih slik v več resolucijah. Struktura je bila zasnovana iz preproste organizacije map, kjer smo slike istih ločljivosti shranjevali v isto mapo in smo slike tako pripravili na nalaganje oziroma obdelavo s strani strežnika. Postopek redčenja slik je sledil metodam poenostavljanja mreže podatkov, saj se posamezna slikovna točka lahko preslika na

posamezno oglišče. V eni sliki, ki je predstavljala najmanjšo enoto podatkov, je zapisanih 10^6 slikovnih točk, 10^3 v vsaki smeri. Če torej vse slikovne točke označimo z indeksi $p_{x,y}$, naša metoda poenostavljanja v posameznem koraku ohrani zgolj sode točke z indeksi $p_{2n,2m};\ n, m \in \mathbb{N} \wedge 0 \leq n \leq \frac{N}{2} \wedge 0 \leq m \leq \frac{N}{2}$, pri čemer z N označimo število slikovnih točk na začetku koraka poenostavljanja. Tako na koncu pridemo do interpolacije točk z metodo najbližjega soseda (angl. nearest neighbour), ki sicer lahko vodi do napak zaradi preredkega vzorčenja (angl. aliasing). To težavo bi lahko reševali z uporabo bilinearne ali bikubične interpolacijske funkcije, ampak to bi povzročilo, da na koncu nobena od 4ih interpoliranih točk ne bi ustrezala prvotnemu stanju, medtem ko z uporabo funkcije najbližjega soseda ohranjamo ustrezne koordinate in nadmorsko višino prikazane točke, ki se na koncu lepo prilegajo na prikazani ortofoto sloj. V dejanski implementaciji so napake zaradi preredkega vzorčenja vidne šele pri najbolj oddaljenih ploščicah mreže terena in niso moteče za uporabniško izkušnjo.

Za boljšo predstavo smo en korak redčenja predstavili na sliki 2.1, obenem pa smo v poglavju o implementaciji napisali več o dobljenih dimenzijah in skupni velikosti slik iz posameznega nivoja na disku.

2.3 Prikaz terena

Jedro te magistrske naloge je prikaz terena v brskalniku na računalniku odjemalca. Za upodabljanje smo uporabljali tehnologije, omenjene v začetku tega poglavja. V naslednjih odstavkih bomo razložili nekaj pojmov, na katerih temelji kasnejši načrt in izvedba naloge.



Slika 2.1: Prikaz koraka poenostavljanja podatkov. Črtkano označene točke bodo odstranjene.

2.3.1 Lepljenje odmikov

Za upodabljanje terena smo izbrali tehniko lepljenja odmikov (angl. displacement mapping) z višinskimi mapami, ki zakodirajo podatke o višini v sliko. Ta tehnika se trudi prikazati odmike oglišč v gosti mreži kot pravo geometrijo. V nasprotju z nekaterimi podobnimi tehnikami, kot je lepljenje izboklin (angl. bump mapping), kjer se skuša odmike prikazati zgolj s spremenjenimi vrednostmi normal, preslikovanje odmikov dejansko prestavi oglišča v določeni smeri in ustvari trikotnike med njimi. Poseben primer preslikovanja odmikov je preslikovanje višine, kjer se odmik uporabi zgolj za spremembo višine oglišča.

Najbolj pogost pristop lepljenja odmikov je branje višine iz posamezne slikovne točke, kjer se ena slikovna točka preslika na točno eno oglišče. Te prebrane vrednosti se kasneje aplicirajo direktno na geometrijo oz. se uporabijo med gradnjo geometrije. Smer odmika h je običajno v primeru terenov navzgor in tako se spremeni vrednost navpične komponente v koordinatah oglišča, kot je prikazano v formuli 2.5. Odmik se lahko izvede tudi v smeri normale, kar se običajno naredi na 3D objektih druge vrste.

$$\mathbf{v} = [v_x \; v_y \; 0]^\top + [0 \; 0 \; h]^\top \tag{2.5}$$

2.4 Nivoji podrobnosti

Ena ključnih komponent učinkovitega prikazovanja terena je primerno prikazovanje vsebine na različnih nivojih natančnosti. Pristop te magistrske naloge se najbolj navezuje na članek [18]. V tem delu bomo natančneje predstavili metodo, opisano v članku, in lastne prijeme, v katerih se naš pristop razlikuje od predstavljenega v članku.

Implementacija metode [18] na začetku organizira višinsko mapo v štiriško drevo, iz katerega se glede na nivo podrobnosti izbirajo ustrezne celice med delovanjem programa. Algoritem izbira take celice, da zagotavlja približno enako razporeditev trikotnikov po zaslonu, ne glede na oddaljenost terena od kamere. Upodabljanje se izvaja na zato namenjenih območjih, ki jih pokrivajo izbrane celice iz štiriškega drevesa, sestavljene v mrežo. Celice za prikaz se izberejo v odvisnosti od razdalje od kamere, zato imajo celice vrednosti, pri katerih so izbrane, preračunane vnaprej. Ker je razmerje v kompleksnosti med vsakim od nivojev približno enako 4, mora biti razmerje razdalj, pri katerih so celice aktivne, približno enako 2, da metoda doseže relativno enakomerno razporeditev trikotnikov na zaslonu. Preračunane razdalje, pri katerih so celice aktivne, se uporabljajo pri izboru celic, ki so prikazane na trenutnem pogledu. Izbor poteka s preletom štiriškega drevesa, od najbolj oddaljenih in najmanj natančnih celic do najbližjih celic, ki so najbolj natančne. Po prehodu čez štiriško drevo tako dobimo celice z vsemi podatki potrebnimi za upodabljanje.

Razlika naše metode od osnovne je najprej vidna v tem, da ne uporabljamo štiriškega drevesa za shranjevanje podatkov, saj nalagamo svoje

višinske mape preko medmrežja. V povezavi s tem moramo mrežo, ki je osnova za upodabljanje na več nivojih podrobnosti, pripraviti prej in glede na položaj mreže potem nalagamo ustrezne podatke.

Osnova za upodabljanje z več nivoji podrobnosti je v pravilni uporabi tehnike preslikovanja odmikov, ki smo jo razložili v prejšnjem delu tega poglavja. Za prikaz terena potrebujemo mrežno geometrijo v ravnini, da lahko ogliščem te geometrije kasneje poiščemo ustrezno višino in jih premaknemo v navpični smeri. Za doseganje višje kvalitete upodabljanja bi radi imeli čimbolj gosto mrežo, da bo gostota oglišč čim večja. Pri tem naletimo na težavo, saj bi uporaba goste mreže za celotno območje Slovenije porabila preveč virov, ker bi s preprostim dodajanjem dodatnih enako velikih ploščic za pokrivanje dvakrat večjega območja porabili dvakrat več ploščic. Tako lahko torej ugotovimo, da je prostorska kompleksnost takega pristopa eksponentna $O(2^n)$, če gledamo število ploščic v odvisnosti od dimenzije območja, ki ga prekrivajo, in bi zato kmalu povzročala neodzivnost aplikacije na večjih območjih. To težavo rešujemo tako, da na območju blizu točke, v katero je usmerjena kamera, uporabljamo manjše goste mreže, okrog katerih dodajamo vedno redkejše mreže. Tako je na začetku 9 najbolj gostih ploščic postavljenih okrog točke v katero je usmerjena kamera, okoli njih je 8 ploščic trikrat večjih dimenzij, ki pa so zaradi tega bolj redke. Tem spet sledi 8 trikrat večjih ploščic itd., dokler mreža ploščic ne pokrije celotnega območja Slovenije. Za boljšo predstavo smo postavitev mreže prikazali na sliki 2.2. S takšnim pristopom zagotovimo logaritemsko prostorsko zahtevnost $O(log\ n)$, saj za prekrivanje območja s trikrat večjimi dimenzijami potrebujemo le za konstantno število dodatnih ploščic.

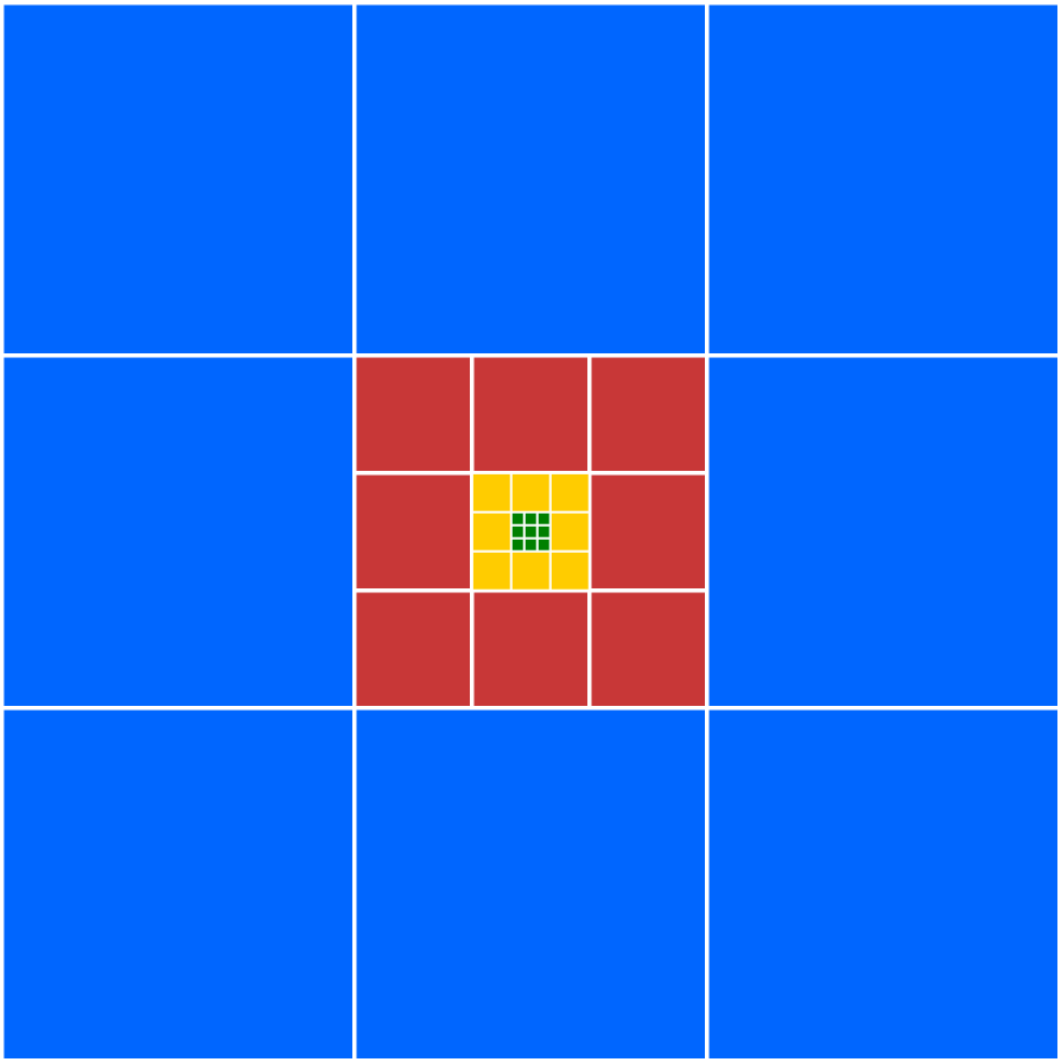
Za konfiguracijo, kjer uporabljamo trikrat večje ploščice na vsakem nivoju, smo se odločili po analizi števila ploščic, ki jih potrebujemo, da pokrijemo območje terena Slovenije. Rezultati analize so prikazani v tabeli 2.1. Pri tem velja omeniti, da bi pri faktorjih večjih od tri potrebovali netrivialne

Tabela 2.1: Analiza števila ploščic glede na dimenzijo pokritega območja za različne faktorje. Dimenzija območja je podana v km.

Faktor 2		Faktor 3	
Dimenzija	**Št. ploščic**	**Dimenzija**	**Št. ploščic**
4	16	3	9
8	28	9	17
16	40	27	25
32	52	81	33
64	64	243	41
128	76	729	49
256	88		
512	100		

začetne konfiguracije z veliko večjim številom ploščic, da bi lahko na zunanjih obročih dodajali ploščice večje za ta faktor, saj ne moremo na primer štirikrat večjih ploščic enostavno postaviti v mrežo okrog ene, štirih ali osmih ploščic.

Za pridobivanje koordinat posameznega oglišča torej potrebujemo ob vsakokratnem ponovnem upodabljanju preračunati več vhodnih podatkov. Naš sistem ploščic z ravninskimi mrežami smo v osnovi zgradili okrog koordinatnega izhodišča in ga kasneje ob izvajanju ponovnega izrisa celotne geometrije zgolj premaknemo na točko, v katero je usmerjena kamera $\mathbf{v}_t$. Posamezna ploščica v sistemu je mreža z $N \times N$ oglišči, velikosti $s \times s$ in ima v začetku definiran odmik svojega levega spodnjega kota od koordinatnega izhodišča o_x in o_y. Koordinate posameznega oglišča $\mathbf{v}$ se torej izračunajo po formuli 2.6. V tej formuli je i zaporedna številka oglišča v smeri x in j zaporedna številka v smeri y. Ne smemo pozabiti na prištevanje višine, katere pridobivanje smo pojasnili v naslednjem poglavju in se jo izračuna po formuli 3.1.



Slika 2.2: Prikaz mreže ploščic na katerih uporabljamo tehniko preslikovanja odmikov.

$$\mathbf{v} = \mathbf{v}_t + [o_x \; o_y \; 0]^\top + s \cdot [\frac{i}{N} \; \frac{j}{N} \; 0]^\top + [0 \; 0 \; h]^\top \tag{2.6}$$

Poglavje 3

Implementacija

V tem poglavju bomo predstavili uporabljene tehnologije in načrt ter izvedbo implementiranega sistema za upodabljanje terena Slovenije. Ogledali si bomo glavne korake, ki so potrebni, da prikažemo teren v brskalniku. Po opisu tehnologij bo govora o obdelavi podatkov in kje jih pridobivamo. Kasneje bomo obdelali upodabljanje in prikaz terena v brskalniku. Na koncu bomo predstavili naš pristop k posodabljanju pogleda pri različnih nivojih podrobnosti.

3.1 Opis tehnologij

Poleg običajnih dvo-dimenzionalnih prikazov slik in drugih vsebin je za temo tega magistrskega dela relevantna predvsem računalniška grafika v treh dimenzijah. Spletni portali se v iskanju čim boljše uporabniške izkušnje vse več poslužujejo 3D vizualizacij, ki so se razvijale skozi različne pristope in tehnologije. Nekaj jih bomo predstavili v naslednjih odstavkih. Ozadje za naslednje opise je pridobljeno iz raziskave [28].

Tabela 3.1: Deklarativni in imperativni pristopi k prikazovanju grafike v brskalniku.

	2D	3D
Deklarativni	SVG	X3DOM, XML3D
Imperativni	`<canvas>`	WebGL

3.1.1 Tehnologije prikazovanja v brskalniku

Prikazovanje vsebine, kjer se proces upodabljanja zgodi na računalniku odjemalca, definiramo kot prikazovanje v brskalniku. To prikazovanje se lahko po [34] deli na imperativno in deklarativno in se poleg tega lahko deli tudi na prikazovanje 2D in 3D vsebine. 2D vsebino lahko na primer vnaprej specificiramo v SVG datoteki, ki bazira na XML standardu, ali jo prikažemo s pomočjo imperativnih ukazov v JavaScriptu, kjer lahko rišemo na element tipa `<canvas>`. Tabela 3.1 prikazuje povzetek omenjene razdelitve.

Razdelitev tehnologij prikazovanja je uporabna predvsem pri obravnavanju upodabljanja 2D grafike, saj lahko enostavno primerjamo podobne primere uporabe. Za razliko je pri upodabljanju 3D grafike težje določiti klasifikacijo - čeprav veliko tehnologij uporablja za prikazovanje imperativne programske jezike, se jih veliko še vedno močno naslanja na uporabo grafa scene, kar bi se lahko smatralo za deklarativni pristop. Avtorji članka [28] so določili, da lahko rečemo za tehnologijo, da je bolj deklarativna kot imperativna, če sledi naslednjim pogojem:

- Je del HTML dokumenta (uporablja besedilni format za deklariranje vsebine brez potrebe po kontekstu).
- Možno jo je integrirati z modelom objektov dokumenta (angl. Document Object Model - DOM).

- Uporablja graf scene.
- Ne dovoli prepisovanja (overloading) v cevovodu upodabljanja.
- Ima visoko stopnjo povezovanja z drugimi platformami.

V članku [28] je nadalje opisanih več tehnologij za prikazovanje 3D grafike v brskalniku. Ker niso relevantne za izdelavo magistrske naloge, jih bomo zgolj našteli v naslednjem seznamu.

- VRML, X3D and ISO standards
- X3DOM
- XML3D
- Xflow
- CSS 3D transforms
- Collada
- Dostop do strojne opreme za grafiko z JavaScriptom
- Stage 3D
- Silverlight
- Java

Z izdelavo magistrske naloge sta najbolj povezana standard WebGL in knjižnica Three.JS, ki ju bomo natančneje opisali v naslednjih podpoglavjih.

3.1.2 WebGL in povezane knjižnice

Khronos, krovna neprofitna organizacija, ki bdi nad standardom WebGL in podobnimi standardi, definira WebGL kot prosto dostopen spletni standard, ki je neodvisen od izbire platforme in predpisuje nizkonivojski 3D vmesnik za namensko programiranje (angl. API), ki je osnovan na standardu OpenGL ES2.0 in izpostavljen preko HTML5 elementa `<canvas>` kot vmesnik v modelu DOM [35]. OpenGL ES (za vgrajene sisteme - angl. Embedded Systems) 2.0 je prirejena verzija standardnega vmesnika OpenGL, namenjena napravam z omejeno močjo procesiranja.

WebGL je namenjen uporabi skupaj z običajnimi spletnimi tehnologijami, saj je stran, ki ga uporablja, še vedno zgrajena iz HTML elementov, medtem ko se 3D komponente spletne strani rišejo preko vmesnika WebGL v JavaScriptu. WebGL je namensko pripravljen, da je optimiziran in dovolj nizkonivojski. Namenjen je izkušenim grafičnim programerjem, ki dobro poznajo bistvene ključne pojme v grafiki, kot je matrično in vektorsko računanje, senčenje in še posebej priporočljivo - organiziranje 3D scene z grafi.

Za ustvarjanje preproste škatle je potrebno ustvariti WebGL kontekst (HTML5 `<canvas>` element), definirati in naložiti kodo senčilnikov za vozlišča in fragmente, pripeti vse enotne podatke (angl. uniforms), pripeti vmesni pomnilnik (angl. buffer) za podatke in na koncu še posredovati dejanske podatke (koordinate vozlišč, podatke o normalah, barvah in o koordinatah teksture). Nazadnje je pred risanjem potrebno nastaviti kamero in perspektivo preko uporabe standardnih matrik modela, pogleda in projekcije. Kot rečeno, velja, da je WebGL namenjen bolj izkušenim programerjem, zato je skupina Khronos pripravila več JavaScript datotek v javno dostopnih predstavitvenih (angl. demo) programih, ki pomagajo razvijati in razhroščevati aplikacije v tehnologiji WebGL.

Obstaja več knjižnic, ki se trudijo programerju olajšati delo v tehnologiji

WebGL, s tem, da njegove notranje konstrukte naredijo bolj abstraktne in s tem omogočijo programerju delo v programskem jeziku višjega nivoja. Te knjižnice so se hitro razširile predvsem zaradi strme krivulje učenja pri tehnologiji WebGL.

Prva od takih knjižnic je bila SpiderGL, ki v svoji originalni različici sestoji iz petih knjižnic: *GL*, v katerem so skrite temeljne funkcionalnosti tehnologije WebGL; *MESH*, ki se uporablja za definicijo in risanje 3D mrež; *ASYNC*, ki je zadolžena za asinhrono nalaganje podatkov; *UI*, s katero rišemo uporabniški vmesnik znotraj GL konteksta; in na koncu *SPACE*, v kateri so številni matematični in geometrijski pripomočki. Kljub temu, da SpiderGL ponuja abstrakcijo številnih WebGL funkcionalnosti, programiranje aplikacij v knjižnici SpiderGL še vedno zahteva znanje 3D konceptov in vsaj osnovno poznavanje standarda OpenGL. Po začetnem obdobju velikega vpliva, je SpiderGL doživel veliko predelav, ki so prinesle nizkonivojske izboljšave, kot na primer možnost razširitev po meri in boljšo integracijo z drugimi knjižnicami.

Med ostalimi knjižnicami je LightGL nizkonivojska ovojnica, ki skrije predvsem funkcionalnosti, ki zahtevajo več programiranja, ampak še vedno zahteva programiranje senčilnikov in manipulacijo matrik. Možno jo je uporabiti skupaj z drugimi knjižnicami, da se ustvari večnivojski vmesnik z več nivoji abstrakcije. OSG.JS je ogrodje za WebGL, ki v razvoju 3D aplikacij poskuša posnemati pristop ogrodja OpenSceneGraph. Druge omembe vredne knjižnice so SceneJS, PhiloGL in GLGE. Zadnja ponuja deklarativne metode za programiranje 3D scene, podobno kot X3DOM ali XML3D.

3.1.3 Three.JS

Verjetno najbolj slavna knjižnica oz. vmesnik za 3D grafiko v spletnih tehnologijah je Three.JS. Čeprav je bila originalno spisana v jeziku ActionScript, je sedaj odprtokodna knjižnica, ki omogoča visokonivojsko programiranje 3D

scen v brskalniku. Njegova modularna struktura omogoča, da se lahko za prikazovanje uporablja več različnih pogonov, kot so WebGL, Canvas in SVG. Knjižnica se razvija s strani več različnih avtorjev, spet zaradi svoje modularne strukture.

Three.JS uporablja graf scene, več različnih tipov kamere in navigacije, nekaj predpripravljenih materialov in senčilnikov, ki jih lahko spišemo tudi sami. Omogoča nalaganje in prikazovanje mreže v več nivojih natančnosti ter animiranje scene preko skeletne animacije ali animacije po vozliščih (angl. morph target animation).

Poleg očitnih sprememb v sintaksi jezika, pristop knjižnice Three.JS ni močno drugačen od pristopa v tehnologiji X3DOM in še posebej XML3D. Material, mreža, kamera in svetloba se morajo dodati na sceno. Mrežo se lahko naloži iz zunanjih datotek preko asinhronih metod nalaganja. Ko se taka datoteka naloži in se ustvari mreža, se mreži določi material in se jo doda na sceno v funkciji povratnega klica. Ta možnost klicanja funkcij je bistvena razlika od prej omenjenih deklarativnih tehnologij. Klicanje funkcij se uporablja vsakič, ko od brskalnika zahtevamo nov okvir (angl. frame) za animacijo in izris scene. Brez tega imperativnega dela bi se objekti vsakič pravilno dodali na sceno, ampak v pogledu se ne bi nič izrisalo, ker se ukazi za izris scene ne bi izvedli.

3.1.4 Risanje grafike na strežniku

V sklopu magistrskega dela je bilo del podatkov potrebno pripraviti na strežniku, zato bomo sedaj opisali še nekaj tehnik, ki so namenjene upodabljanju grafike s pomočje virov na strežniškem računalniku. Snov je spet povzeta iz članka [28].

Pristop oddaljenega upodabljanja grafike običajno vključuje generira-

nje podatkov na strežniku, ki nato te podatke posreduje odjemalcu, da jih prikaže. Večino raziskav v tem polju ni neposredno povezanih s spletnimi tehnologijami, predvsem zaradi dejstva, da običajen odjemalec podatkov iz strežnika ni zasnovan kot spletna aplikacija. Iz vidika raziskovanja je možno oddaljeno upodabljanje v grobem razdeliti na tri dele: grafični ukazi, slikovne točke in vektorji. Omenimo naj tudi proces poenostavljanja in delitve 3D objektov in drugih podatkov, o tej temi je bilo več govora v poglavju 2.2.

Pri grafičnih ukazih so nizkonivojski klici strežniške GPE prestreženi in posredovani odjemalcu, ki nato izriše in prikaže rezultat teh klicev.

Pri metodi slikovnih točk strežnik izriše celotno sliko in jo posreduje odjemalcu zgolj v prikaz. Tak pristop se lahko prevede na problem prenašanja podatkov, ker je zelo primerljiv pretočnemu predvajanju videoposnetkov. Pri temu pristopu lahko uvedemo več optimizacij, kot na primer sinhronizacija bolj kvalitetnega izrisa na strežniku z manj kvalitetnim izrisom na odjemalcu, selektiven prenos slikovnih točk ali optimizacija kodiranja prenešenih slik.

V načinu pridobivanja vektorjev za posredovanje odjemalcu se na strežniku uporabljajo različne tehnike odkrivanja oblik (angl. feature exctraction), tako v 2D kot tudi v 3D. Prednost te metode je, da lahko 3D objekte iz posredovanih vektorskih podatkov izrisujejo tudi odjemalci, ki nimajo sposobnosti izvajanja 3D operacij, saj se zahtevne 3D transformacije izvedejo na strežniku.

Na koncu velja omeniti, da obstajajo prizadevanja za združevanje nekaterih od omenjenih tehnik. Nekateri pristopi zajamejo navodila za upodabljanje med izvajanjem aplikacije, jih izvedejo in nato pošljejo spremembe v sceni odjemalcu, ki jih izriše na svoji grafično procesni enoti. Če odjemalec ni sposoben izrisati scene, sceno izriše strežnik in odjemalcu pošlje pretočno video vsebino.

3.2 Pripravljanje podatkov

V aplikaciji uporabljamo dve vrsti podatkov. Najprej na podlagi podatkov iz digitalnega modela reliefa določamo lastnosti geometrije, ki jo izrisujemo. DMR podatki obsegajo točke v novem koordinatnem sistemu Slovenije D96/TM in nadmorsko višino vsake podane točke. Druga vrsta podatkov so slike v ortofoto sloju, ki jih želimo prikazati kot teksture na vrhu naše geometrije. Obe vrsti podatkov pridobivamo iz portalov, ki jih upravlja ARSO.

3.2.1 DMR podatki

O DMR podatkih, ki jih ARSO ponuja na svojem portalu [36], smo govorili že v delu 2.1.1. Sedaj bomo več besed namenili konkretni obliki podatkov, njihovi obdelavi in shranjevanju.

Kot omenjeno v delu 2.1.1, je DMR, ki ga prikazujemo, mreža gostote 1 x 1 m, ki ima v vsakem vozlišču točko s podanimi koordinatami in nadmorsko višino. Na portalu LIDAR [36] je območje Slovenije razdeljeno na bloke v velikosti 1 km^2. DMR podatke lahko prenašamo po delih in sicer lahko s klikom na posamezen blok prenesemo besedilno datoteko, v kateri vsaka vrstica predstavlja eno točko v mreži digitalnega modela reliefa. Ker so bloki velikosti 1000 x 1000 m, imajo datoteke po 10^6 vrstic, razen nekaterih manjših datotek ob državni meji. Primer nekaj podanih točk si lahko ogledamo v izseku 1, kjer je nekaj podatkov iz bloka D96/TM s koordinatami x: 455000 in y: 89000.

Takoj smo ugotovili, da je tak zapis zelo neučinkovit, ker delamo zgolj s števili in bi jih zagotovo lahko zapisali v binarni obliki ter da zaradi lepo podane mreže potrebujemo zgolj podatke o višini, ker podatke o koordinati

Algoritem 1 Nekaj točk iz digitalnega modela reliefa v besedilni obliki. Prvi dve števili sta podatka o koordinati, zadnje število je nadmorska višina.

```
 1: 455000.00;89000.00;520.19
 2: 455000.00;89001.00;520.19
 3: 455000.00;89002.00;519.73
 4: ...
 5: 455633.00;89796.00;350.70
 6: 455633.00;89797.00;350.54
 7: 455633.00;89798.00;350.35
 8: ...
 9: 455999.00;89997.00;456.65
10: 455999.00;89998.00;456.65
11: 455999.00;89999.00;457.99
```

lahko pridobimo iz vrstnega reda podatkov. Najprej smo podatke o višini pomnožili s 100, da smo lahko delali s celimi števili. Nato smo preračunali, da za doseganje največje natančnosti, ki jo dani podatki omogočajo, potrebujemo 19 bitov, ker potrebujemo zapisati vrednosti od 0 in vse do 286400. Za kodiranje posameznega podatka o višini, zapisanega z 19 biti, smo zasnovali preprosto shemo, kjer smo 8 najmanj pomembnih bitov zapisali kot modro barvo, naslednjih 8 bitov kot zeleno in zadnje 3 bite kot rdečo barvo. V tem pristopu se je pojavila težava v ostrih prehodih med barvami, ker so se na teh mestih pri upodabljanju terena začeli pojavljati razni artefakti v obliki lukenj in štrlečih konic. Več o razlogih za te pojave in postopku reševanja v delu 3.2.3.

Najprej smo poskušali števila zapisati v preprosti binarni obliki, pri čemer smo morali biti pozorni pri zapisovanju binarnih števil, predvsem zaradi postavitve pomembnih bitov (angl. 'little vs. big endian'). Uporabljeni programski jeziki so namreč uporabljali različne dogovore glede zapisovanja in branja binarnih števil, zato smo morali najprej najti pravilno shemo kodiranja in dekodiranja števil, preden je bilo takšno zapisovanje uporabno. Zapis števil v binarni obliki je skrčil podatke na 14 % njihove prvotne velikosti. Podatki v binarnem zapisu so bili neugodni za dekodiranje v brskalniku,

saj so zahtevali kompleksno dekodirno shemo, zato smo na koncu za kodiranje podatkov izbrali zapis v barvnih kanalih png slik, ki smo jih tudi lažje obdelovali na strežniku. Tako zapisani podatki so na koncu obsegali 5 % prvotne velikosti. Podatke o višini smo torej zapisali v slikovne točke slike z dimenzijami 1000 x 1000 slikovnih točk. Praktična implementacija je sledila postopku opisanem v delu 2.2.

3.2.2 Ortofoto sloj

Na izrisani geometriji prikazujemo preprost ortofoto sloj, na katerem bodo slike terena v različnih ločljivostih, ki jih bo narekoval ustrezni nivo natančnosti. Prikazane slike bodo namenjene lažji orientaciji po terenu in popestritvi prikazane geometrije. Pridobivali jih bomo prek spletnih storitev, ki jih je pripravil ARSO. Ta strežnik sprejema zahteve za posamezne slike v drugem koordinatnem sistemu, kot ga uporabljamo za prikaz geometrije, zato bo ob pošiljanju zahtevkov potrebna transformacija. Več o tem v poglavju o implementaciji.

Sistem je zasnovan tako, da bi brez večjih težav na geometriji terena prikazali tudi slike drugih slojev, ki se jih da pridobivati preko spletnih storitev. Tako bi lahko na terenu prikazali razne zemljevide, geološke karte, rastlinske zemljevide in še veliko drugih slojev.

3.2.3 Obdelava in shranjevanje podatkov

Podatke smo morali najprej prenesti iz spletnih storitev, ki jih ponuja ARSO. Za prenašanje smo uporabili preprosto skripto, ki je s programom `wget` prenesla `.asc` datoteko z višinskimi podatki za posamezen km^2, poklicala program za kodiranje podatkov v sliko in na koncu zbrisala datoteko `.asc`. URL naslove za prenašanje datotek smo morali zgraditi iz bloka, v katerem se je

datoteka nahajala, in iz začetnih koordinat podatkov v datoteki[1].

Program za kodiranje podatkov smo implementirali v programskem jeziku `python`. Za omenjeni jezik smo se odločili zaradi programskih paketov, s katerimi smo si lahko olajšali delo in tako pohitrili razvoj aplikacije, čeprav smo s tem nekoliko žrtvovali hitrost izvajanja kodiranja. Naloga programa za kodiranje je bila prebrati podatke o višini iz besedilne `.asc` datoteke in jih zapisati v slikovne točke ter na koncu shraniti kot sliko. V zadnjem koraku smo dodali še redčenje slike in shranjevanje slik manjših ločljivosti. Celoten postopek je prikazan v psevdokodi algoritma 2.

Po kodiranju vseh blokov smo dobili drevesno strukturo, ki je omogočala osnovno razdelitev za nalaganje podatkov pri več nivojih natančnosti, saj smo v isto mapo shranjevali slike v isti ločljivosti. Najprej nas je skrbelo, da se bo zaradi toliko nivojev shranjevanja slik občutno povečal potreben prostor na disku za njihovo shranjevanje, ampak kmalu smo po nekaj izračunih ugotovili, da s tem ne bo težav, ker se velikost posamezne slike zmanjšuje eksponentno. V tabeli 3.2 so prikazane dimenzije in skupna velikost vseh slik na posameznem nivoju, ki smo jih pridobili po omenjenem zmanjševanju.

Kot omenjeno v začetnem opisu implementacije poenostavljanja podatkov 3.2.1, smo ob poskusih skaliranja slik za zmanjšanje količine prenešenih podatkov naleteli na težave z dekodiranjem slikovnih točk ob ostrih robovih med posameznimi barvami na sliki s podatki o višini. Algoritmi za skaliranje slik namreč predpostavljajo, da so posamezni barvni kanali slikovne točke neodvisni in jih tako tudi interpolirajo. V našem primeru, ko združujemo podatke iz kanalov v skupen podatek o višini, to ni držalo, zato smo morali opustiti vsakršno skaliranje slik in uporabiti pravilne dimenzije slik, ki smo jih pripravili med poenostavljanjem podatkov.

[1]Primer enega od URL naslovov:
`http://gis.arso.gov.si/lidar/dmr1/b_23/D96TM/TM1_500_157.asc`

Algoritem 2 Psevdokoda programa za kodiranje podatkov.

```
DIMENZIJA_BLOKA ← 1000
x ← parameter[0]
y ← parameter[1]
asc_datoteka ← preberi csv datoteko(x, y)
biti ← seznam()
for i ← 0 to i < DIMENZIJA_BLOKA do
    for j ← 0 to j < DIMENZIJA_BLOKA do
        xyz ← beri csv vrstico(asc_datoteka)
        z ← xyz[2]
        z ← int(float(z) × 100)
        pripni seznamu( biti, logični in( logični pomik desno( z, 16 ), 255 ) )
        pripni seznamu( biti, logični in( logični pomik desno( z, 8 ), 255 ) )
        pripni seznamu( biti, logični in( z, 255 ) )
    end for
end for
slika ← preoblikuj v matriko(biti, (block_dimension, block_dimension, 3))
rotiraj za 90 stopinj(slika)
nivo ← 10
while nivo > 1 do
    shrani png sliko(slika, nivo, xname, yname)
    odstrani sode slikovne točke(slika)
    nivo ← nivo - 1
end while
```

Tabela 3.2: Dimenzije in skupna velikost slik na posameznem nivoju.

Nivo	Dimenzija	Skupna velikost vseh slik
10	1000 x 1000	22.558 MB
9	500 x 500	6.421 MB
8	250 x 250	1.773 MB
7	125 x 125	489 MB
6	63 x 63	139 MB
5	32 x 32	41 MB
4	16 x 16	13 MB
3	8 x 8	4,9 MB
2	4 x 4	2,4 MB
1	2 x 2	1,6 MB

Ortofoto sloj smo pridobili iz strežnikov v lasti ARSO, na način, ki smo ga opisali že v oddelku 3.2.2. Pri pridobivanju je bilo potrebno le paziti na uporabljanje pravilnih koordinat. Te smo pridobili s pretvorbo koordinat na geometriji, ki so bile v sistemu D96/TM, v sistem D48/GK, ki ga uporablja ARSO strežnik. Za pretvorbo smo uporabili knjižnico Proj4js [37] in konfiguracijske podatke iz strani spatialreference.org [38].

3.3 Prikaz terena v brskalniku

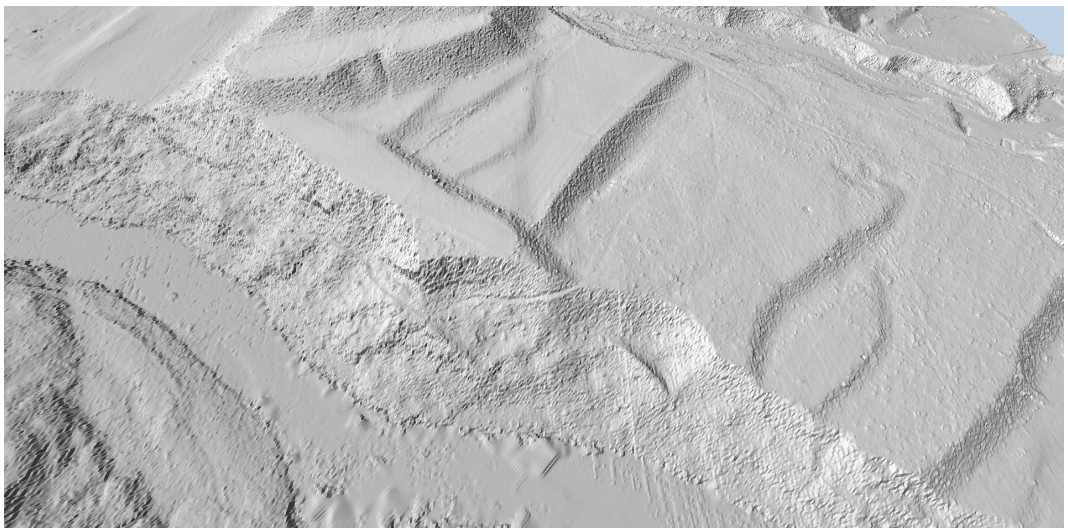
Opisali bomo dva temeljno različna pristopa, ki smo ju imeli na izbiro pri načrtovanju upodabljanja geometrije in razloge, zakaj smo izbrali enega od njiju. V zadnjem delu bomo napisali še nekaj besed o načrtovanju prikaza ortofoto sloja.

3.3.1 Geometrija v CPE

Najprej smo v želji po doseganju najvišje možne natančnosti prikaza risali geometrijo s programom na procesorju. Podatke o višini smo spet razbrali iz slike, v katere so bili zakodirani, in nato smo po vrsti obdelali po 4 sosednje točke in zanje narisali po 2 trikotnika. Tako je nastala zelo točno prikazana geometrija, ki jo lahko vidite na slikah 3.1 in 3.2. Kljub temu točnemu prikazu je bila slika od blizu vizualno neprivlačna, saj so posamezni trikotniki, ki gradijo geometrijo, preveč izstopali zaradi ostrih robov na geometriji. Ti robovi se pri implementaciji lepljenja odmikov na GPE niso pojavljali zaradi interpolacije koordinat prikazanih slikovnih točk med oglišči geometrije [39]. Poleg tega je tak prikaz učinkovito deloval le na omejenem območju v premeru nekaj km zaradi prevelike količine podatkov. Ob prvih poskusih nalaganja večjih območij je grajenje geometrije zahtevalo preveč časa ter pomnilnika. Kljub odpravljanju teh težav je bil ta pristop zelo neučinkovit zaradi prevelike porabe virov in je povzročal slabo uporabniško izkušnjo, saj je uporabnik dolgo časa čakal na nezadovoljiv prikaz. Natančnejša analiza porabe virov pri grajenju geometrije je opisana v 2.4.

Kodo za dekodiranje podatkov na CPE smo spisali v jeziku JavaScript in je v svojem bistvu le obraten postopek dekodiranju, ki smo ga predstavili v psevdokodi, prikazani v algoritmu 2. Pri tem pristopu smo sliko s podatki o višini izrisali na `<canvas>` element v brskalniku, prebrali njene slikovne točke in na koncu iz posameznih barvnih kanalov z manipulacijo bitov ter nekaj logičnimi operacijami dekodirali podatke o višini. Te podatke smo shranili v preprost seznam, ki smo ga kasneje uporabili za izgradnjo geometrije.

Za gradnjo geometrije v CPE smo se v veliki meri naslonili na knjižnico Three.JS in njene konstrukte. Osnoven načrt risanja je bil predstavljen v prvem odstavku tega dela. Geometrijo smo implementirali kot generično geometrijo, kateri smo sami specificirali vse trikotnike, ki jih je bilo potrebno izrisati. Za vsa vozlišča trikotnika je bilo potrebno preračunati koordinate,



Slika 3.1: Geometrija zgrajena na CPE.

Slika 3.2: Prejšnji geometriji je dodan ortofoto sloj. Prikazan je levi breg Save blizu Radovljice.

smer normal, barvo glede na nadmorsko višino in teksturne uv koordinate glede na položaj v celotni geometriji. Na koncu smo zgrajeni geometriji dodali še enega od vgrajenih materialov, kateremu je bila pripeta tekstura ortofoto sloja, ki smo ga prikazali na geometriji.

Zaradi številnih naštetih preračunavanj v CPE se je pri omenjenem pristopu izkazalo, da je zelo neučinkovit, saj je grajenje geometrije ob vsakem posodabljanju terena trajalo več 10 s. Obenem je ob tem preračunavanju aplikacija v brskalniku postala povsem neodzivna, kar je močno poslabšalo uporabniško izkušnjo. Poleg naštetih pomanjkljivosti pristopa s CPE je bilo tudi samo prikazovanje geometrije precej zahtevno za grafiko, saj smo ob geometriji zgrajeni v največji natančnosti lahko prikazovali le okrog 10 sličic na sekundo. Ti razlogi so nas prepričali, da smo preizkusili podatke dekodirati in upodabljati v GPE.

3.3.2 Geometrija v GPE

Zaradi prej navedenih razlogov smo se obrnili na druge metode. Pri glavnem delu implementacije magistrskega dela smo se v veliki meri zgledovali po metodi [18], ki smo jo za naše potrebe nekoliko modificirali. Uporabili smo osnovno idejo, da geometrijo prikazanega terena prikažemo z metodo lepljenja odmikov, ki je natančneje opisana v članku [40]. V grobem gre za pristop, kjer najprej na sceno postavimo mrežo v ravnini. Kasneje v senčilnike za vozlišča naložimo podatke o višini v obliki teksture (slike) in iz te teksture razberemo podatek o višini na tistem mestu. Vozlišče nato za prebrano vrednost premaknemo v vertikalni smeri. V našem primeru je dekodiranje višine sledilo isti shemi kot kodiranje. Iz modre komponente barve preberemo najmanj pomembne bite podatka o višini, iz zelene komponente preberemo 8 srednjih bitov in iz rdeče komponente upoštevamo zgolj 3 preostale, ampak najbolj pomembne bite.

Pri upodabljanju na GPE smo za osnovne elemente upodabljanja, kot so osvetlitev, upravljanje s kamero in pregled nad prikazanimi objekti, še vedno uporabili knjižnico Three.JS. Razlika je bila v tem, da smo dekodiranje podatkov o višini, grajenje geometrije in prikazovanje ortofoto sloja implementirali s programom na GPE. Okvirno je pristop opisan v prejšnjem odstavku, sedaj bi radi dodali še nekaj podrobnosti.

V prvem koraku, kjer smo ustvarjali mrežo ravninskih ploščic, ki smo jih kasneje uporabili na GPU za upodabljanje, velja omeniti posebnost, da smo za vse ploščice uporabili zgolj en Three.JS objekt ravninske geometrije, kar močno zmanjša porabo pomnilnika in drugih virov. To smo dosegli tako, da smo enemu objektu geometrije, ki je imel položaj nastavljen na koordinatno izhodišče, pri ustvarjanju objektov mreže (angl. mesh) zmeraj pripeli drug objekt materiala. V teh različnih objektih materiala smo specificirali programe za upodabljanje na GPE (senčilniki) in podatke, ki jih te programi uporabljajo.

Uporabo ene same geometrije za vse plošče v ravnini je omogočal senčilnik vozlišč, kjer smo vsako oglišče pred izrisom najprej premaknili na prave koordinate s pomočjo vnaprej določenega globalnega odmika in dinamično prebranega odmika na račun trenutnega pogleda kamere. Po premiku na prave vodoravne koordinate smo za vsako vozlišče prebrali podatek o višini iz teksture, kamor smo naložili slike s temi podatki, in nato smo premaknili vozlišče za prebrano vrednost v navpični smeri navzgor. Pristopi k preslikovanju odmikov na GPE uporabljajo funkcionalnost senčilnikov, ki omogoča pridobivanje vrednosti teksture na lokaciji oglišča. To funkcionalnost je vpeljal Shader Model 3.0.

V naši implementaciji lepljenja odmikov smo preizkusili CPE pristop kot tudi GPE pristop in se na koncu odločili za slednjega. Pri CPE pristopu je obdelava posameznih slikovnih točk za pridobivanje podatkov pripeljala

do zahtevnosti $O(n^2)$, ker smo morali brati višino vseh vozlišč v enem procesu, medtem ko se na GPE breme branja podatkov o višini razporedi na več procesorjev. Pri obdelovani količini podatkov - število oglišč je na enem kvadratnem kilometru lahko segalo do 10^6 - je bila neučinkovitost CPE pristopa očitna. Usmerili smo se v GPE pristop, kjer smo iz slike, v kateri so bili zakodirani podatki o višini, najprej pridobili vektor štirih komponent. Vektor je vseboval vrednosti med 0 in 1, prve tri za barvne kanale rdeče, zelene in modre ter zadnje število za opis prosojnosti. Končen podatek o višini za obravnavano oglišče smo torej pridobili s skalarnim produktom v formuli 3.1. Koeficiente smo pridobili tako, da smo interval med 0 in 1 najprej skalirali do 255, saj smo na tem intervalu zapisovali števila med kodiranjem. Vsakemu od barvnih kanalov smo dodali primeren faktor v obliki potence števila 2, kar je služilo namenu premika bitov na pravilno mesto. V formuli 3.1 spremenljivka k predstavlja koeficient, s katerim reguliramo prebrano višino, vrednost spremenljivke $\mathbf{b}$ pa je vektor barvnih vrednosti iz teksture, opisan prej v tem odstavku.

$$h = k \cdot (\mathbf{b} \bullet [(255 \cdot 2^{16})\ (255 \cdot 2^{8})\ 255\ 0]^{\top}) \tag{3.1}$$

Tako pridobljeno mrežo ploščic smo prikazali na takih koordinatah, da so se lokacije na prikazani geometriji ujemale s koordinatami lokacij na realnem terenu. Med drugim smo za lažje delo s koordinatami za smer navzgor izbrali koordinatno os Z, čeprav je to v privzeti konfiguraciji os Y. Tako smo si omogočili lažje prikazovanje ortofoto sloja in pripravili teren za druge razširitve sistema. Pri tem smo naleteli zgolj na eno oviro in sicer je avtomatsko izločanje objektov iz vidnega polja (angl. frustum culling), implementirano na CPE znotraj knjižnice Three.JS, delovalo narobe, ker so bile, gledano iz CPE, vse prikazane mreže postavljene v koordinatno izhodišče, naš teren pa se je s pomočjo odmikov na GPE prikazoval na realnih koordinatah Slovenije, kar je daleč stran od koordinatnega izhodišča. Kamera, ki je bila locirana nad prikazanim terenom, je bila torej, gledano iz vidika

CPE, postavljena popolnoma drugje kot pa mreže, na katerih smo prikazali teren. Tako je bilo izločanje objektov iz vidnega polja krivo, da so prikazane mreže ob naključnih pogledih preprosto izginile, ker je program upodabljanja v CPE domneval, da so izven pogleda. Koordinat prikazanih mrež nismo mogli popravljati tudi na CPE, ker bi s tem pokvarili logiko drugih delov sistema. Iz teh razlogov smo izločanje objektov iz vidnega polja onemogočili, to pa nam presenetljivo ni prineslo bistvenega poslabšanja v učinkovitosti prikazovanja, saj se število prikazanih sličic na sekundo ni zmanjšalo.

3.3.3 Prikazovanje ortofoto sloja

Nazadnje smo pri prikazu načrtovali prikaz ortofoto sloja na izrisani geometriji. O pridobivanju podatkov smo več napisali v odlomku 3.2.2. V poskusu grajenja geometrije v CPE smo morali tudi sami določati koordinate znotraj teksture, ki smo jo prikazali na geometriji. Tudi iz tega razloga smo se odločili za upodabljanje v senčilnikih, ker je v tem pristopu celoten senčnilnik fragmentov namenjen barvanju geometrije in tudi računanje koordinat postane zelo enostavno zaradi možnosti samodejnega določanja teksturnih uv koordinat in dobre podpore za operacije na vektorjih. Pri določanju barve smo tako v senčilniku za vozlišča določili pravilno vrednost teksturnih uv koordinat. Te koordinate so se nato interpolirale za posamezne slikovne točke in na podlagi teh koordinat smo razbrali barvo iz teksture s sliko ortofoto posnetka ter s to barvo pobarvali posamezno slikovno točko.

Spletna storitev iz katere prenašamo ortofoto slike, podpira koordinate v specifičnem koordinatnem sistemu, zato smo vedeli, da bomo morali koordinate naše geometrije preprojecirati preden bomo klicali omenjene spletne storitve. Zaradi nalaganja slik v več nivojih natančnosti smo uporabili še nekaj prijemov, ampak več o tem v kasnejših oddelkih.

3.4 Nivoji natančnosti in posodabljanje pogleda

Za omogočanje upodabljanja terena na različnih nivojih natančnosti smo se poslužili nekaterih prijemov iz metode [18]. Za izhodišče smo si izbrali delno implementacijo te metode, ki smo jo našli v repozitoriju [41]. Najprej smo teren razdelili na kvadrate, ki se povečujejo sorazmerno z oddaljenostjo od središča pogleda. Postavitev je prikazana na sliki 2.2. Zmanjševanje ločljivosti z večanjem razdalje od lokacije pogleda dosežemo s tem, da v kvadratih prikazujemo mrežo z vedno isto gostoto. Ker so zunanje mreže razpete nad večjim območjem, upodabljajo teren v nižji ločljivosti. Tak pristop deluje tako pri upodabljanju terena kot tudi pri prikazu ortofoto sloja.

Ob obisku spletne strani se med zagonom najprej naložijo zgolj grobi podatki, ki pokrijejo celo Slovenijo. Ključen korak v delovanju aplikacije sledi, ko uporabnik začne premikati kamero. Ob zadostnem premiku v navpični ali vodoravni smeri se določi nova lokacija opazovanja. Kvadrate, ki delijo teren, premaknemo tako, da razdelijo teren okrog te opazovane točke in na novo se naložijo vse višinske mape in ortofoto posnetki.

Eden od ključnih elementov sistema, ki nam je omogočil učinkovito prikazovanje terena, je seveda upodabljanje danih podatkov z različnimi nivoji podrobnosti. Pri upodabljanju na CPE smo v ta namen uporabili zgolj različne ločljivosti slik s podatki o višini, brez da bi jih posebej obdelali. Ta pristop je bil zelo okoren in nam je povzročal precej težav, tudi ko smo ga poskušali uporabiti pri upodabljanju na GPE. Tako smo na koncu prišli do pristopa, ki smo ga bolj splošno opisali v prejšnjih delih tega poglavja, natančneje pa ga bomo opisali v naslednjih odstavkih.

Sistem upodabljanja na večih nivojih natančnosti je močno povezan s potekom dogajanja, ki ga določa uporabnik z navigacijo po terenu. Potek je

orisan v algoritmu 3. Ob začetku obiska spletne strani se uporabniku prikaže območje cele Slovenije iz ptičje perspektive. Ob vzpostavitvi sistema se ustvarita dve mreži za upodabljanja terena. Prva ima zelo grobo geometrijo in se uporablja za začasni prikaz med nalaganjem in posodabljanjem druge, natančnejše mreže. Medtem ko je groba mreža homogena in pokriva celo Slovenijo, je pri natančnejši mreži razporeditev posameznih ploščic namenjena zagotavljanju več nivojev podrobnosti. Razporeditev smo natančneje opisali v poglavju 2.4.

Po tem začetnem izrisu terena je potrebno teren posodabljati z novimi podatki, saj je potrebno vedno prikazati najbolj točne podatke okrog tarče pogleda, ki pa se spreminja ob vsakem premiku. Najprej smo nalaganje novih podatkov prožili ob vsakem premiku ampak smo hitro ugotovili, da to močno poslabša uporabniško izkušnjo zaradi veliko čakanja ob pogostem nalaganju sistema. Na podlagi teh razlogov smo po naložitvi podatkov dovolili premikanje pogleda po terenu za nekaj kilometrov, vse dokler je natančnost dovolj visoka, šele nato se proži ponovno nalaganje. Velikost območja, po katerem se lahko premikamo brez ponovnega nalaganja, je odvisna od višine, pri kateri opazujemo teren.

Nove podatke o višini in slike ortofoto sloja pridobivamo za vsako ploščico mreže posebej in zato smo morali razviti strežniško aplikacijo, ki servira slike s podatki o višini prilagojene za vsako enoto v mreži. Podatke namreč posodabljamo tako, da v aplikaciji odjemalca za vsako enoto natančne mreže pošljemo na strežniško aplikacijo zahtevek za sliko izbrane ločljivosti in velikosti, ki sta odvisni od koordinat in velikosti te enote. Strežniška aplikacija prebere minimalne in maksimalne koordinate ter nivo natančnosti, nato si v spomin naloži dovolj predpripravljenih slik izbrane ločljivosti, da lahko z njimi prekrije zahtevano območje in jih na koncu obreže, da se prilegajo zahtevanemu območju. Ko se k odjemalcu prenese obrezana slika s podatki o višini za eno enoto, se za isto enoto prenese tudi slika ortofoto sloja, ki je

prav tako prilagojena območju obravnavane enote.

Algoritem 3 Oris dogajanja med uporabo vmesnika.

```
grobi_podatki ← naloži višinske podatke za celo Slovenijo()
grobe_ortofoto_slike ← naloži ortofoto slike za celo Slovenijo()
groba_mreža ← inicializiraj mrežo(grobi_podatki, grobe_ortofoto_slike)
pokaži(groba_mreža)
natančna_mreža ← zgradi natančno mrežo()
while TRUE do
    if premik je večji od dovoljenega then
        skrij(natančna_mreža)
        pokaži(groba_mreža)
        popravi lokacijo natančne mreže
        for i ← 0 to i < število ploščic v natančni mreži do
            natančni_podatki ← naloži natančne višinske podatke za ploščico
            natančna_ortofoto_slika ← naloži natančno ortofoto sliko za ploščico
            ploščica→posodobi(natančni_podatki, natančna_ortofoto_slika)
        end for
        pokaži(natančna_mreža)
        skrij(groba_mreža)
    end if
    izriši teren
end while
```

Poglavje 4

Analiza implementacije

Pri analizi implementirane metode upodabljanja z večimi nivoji natančnosti bomo primerjali metodo upodabljanja na GPE s predelano metodo upodabljanja na CPE. Predelali smo jo tako, da smo teren razdelili na enako velike dele ter na vsakem prikazali en LOD objekt knjižnice Three.JS. V tem objektu je že implementirano preverjanje razdalje od kamere in na podlagi te razdalje se lahko prikažejo različno natančne geometrije za upodabljanje terena.

Analize smo poganjali vedno na istem odjemalcu in aplikacije smo stregli iz strežnika, kjer smo v istem direktoriju zgolj menjali kodo za različne metode, podatki so ostajali vedno isti. Točne specifikacije odjemalca in strežnika smo predstavili v tabeli 4.1. Uporabljali smo brskalnik Google Chrome, različica 60.0.3112.101 (64 bitov) na sveže zagnanem odjemalcu z operacijskim sistemom Windows 10 Pro (64 bitna različica), ki je imel tako na voljo kar največ razpoložljivih virov.

Metode smo primerjali glede na njihov začetni čas nalaganja in čas posodabljanja pogleda, učinkovitost prikazovanja smo primerjali na podlagi hitrosti upodabljanja, izmerjene s številom sličic na sekundo, na koncu pa smo preverili še porabo pomnilnika pri delovanju aplikacije in količino prenesenih

Tabela 4.1: Specifikacije odjemalca in strežnika.

	Odjemalec	Strežnik
CPE	Intel(R) Core(TM) i3-2330M @ 2,20 GHz	Intel(R) Xeon(R) CPU L5520 @ 2.27GHz
GPE	Nvidia GeForce(R) GT 540M CUDA(TM) 2 GB	ATI ES1000
RAM	8096 MB	5953 MB
OS	Windows 10 Pro 64 bit	Ubuntu 16.04

podatkov.

Obe metodi smo za vsakega od parametrov pognali večkrat, da smo lahko dobljene rezultate povprečili in s tem nekoliko izločili človeški faktor, ki je bil prisoten pri merjenju. Vsakega od parametrov smo merili ob zaključku začetnega nalaganja in po treh premikih, ki so zahtevali polno nalaganje podatkov.

4.1 Analiza hitrosti upodabljanja

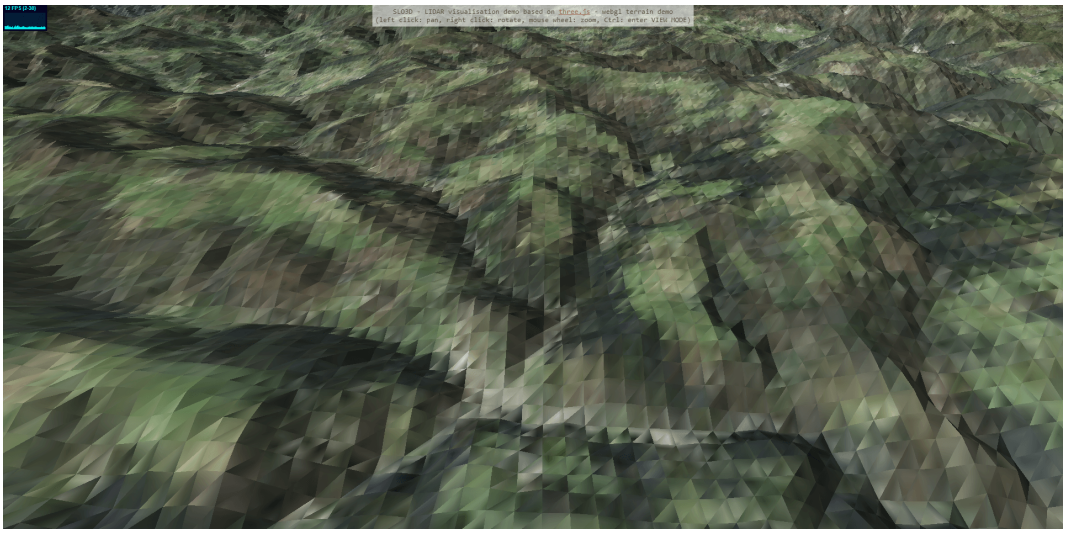
Prvi in najpomembnejši parameter, ki smo ga merili, je hitrost upodabljanja 3D objektov, ki jih prikazujemo na sceni. Meritev smo izvedli s pomočjo knjižnice stats.js [42], ki nam je prikazovala število prikazanih sličic na sekundo. Kot rečeno v prejšnjih delih tega poglavja, smo dobljene meritve zajemali na sveže naloženem sistemu in kasneje sistemu ob nekaj nalaganjih podrobnih podatkov. Pri drugi vrsti meritve, pri pregledu podrobnih podatkov, smo vedno pogled približali do površja terena in ga usmerili proti obzorju, kot je prikazano na sliki 4.1. Na ta način smo preizkusili kakovost prikazovanja terena na večih nivojih natančnosti. Vrednosti pridobljenih meritev so podane v tabeli 4.2.

Pri hitrosti upodabljanja smo ugotovili, da je v obeh primerih bolj učinkovita metoda upodabljanja na GPE. Razloge gre iskati v dobri razporeditvi računskih

Slika 4.1: Posnetek vmesnika aplikacije s pogledom proti obzorju. Grajski grič v Ljubljani in Julijske Alpe s Triglavom v ozadju.

Tabela 4.2: Meritve hitrosti prikazovanja. Navedeno je število prikazanih sličic na sekundo (angl. frames per second). Višje je boljše.

	GPE	CPE
Začetni pogled	**57,3**	33,2
Po treh premikih	**47,2**	18,4



Slika 4.2: Posnetek vmesnika aplikacije implementirane na CPE. Prikazana je okolica kraja Sovodenj.

operacij, ki jo lahko dosežemo s programiranjem senčilnikov vozlišč in fragmentov. Te operacije se razporedijo po več procesorjih, ki jih ima na voljo GPE. Hitrost upodabljanja se zaradi nespremenjene gostote mreže med upodabljanjem na GPE niti ne zmanjšuje, medtem ko se pri upodabljanju na CPE gostota mreže bistveno spreminja in to konkretno vpliva na upad hitrosti upodabljanja po nekaj premikih in približanemu pogledu. Argument v korist upodabljanju na GPE je še bolj podkrepljen ob dejstvu, da metoda upodabljanja na GPE prikaže lepšo sliko, kar se zgodi predvsem na račun več prenesenih podatkov. Primerjava upodabljanja je prikazana na slikah 4.2 in 4.3.

4.2 Analiza omrežne aktivnosti

Pri analizi omrežne aktivnosti smo najprej analizirali količino prenešenih podatkov. Pri merjenju tega parametra je bilo v večini primerov nesmiselno

Slika 4.3: Posnetek vmesnika aplikacije implementirane na GPE. Prikazana je okolica kraja Sovodenj.

ponavljati meritve in jih kasneje povprečiti, ker je bilo za večino pogledov potrebno prenesti vedno isto količino podatkov. Podatki so se v posameznih meritvah razlikovali le, ko se je pri premikanju pogleda nalagalo drugo območje na terenu.

V meritev so bili vključeni vsi prenešeni podatki - slike z višinskimi podatki, slike za teksture ortofoto sloja in koda za izvajanje programa. Količino podatkov smo izmerili s pomočjo orodij za razvijalce v brskalniku Google Chrome, ki v zavihku za omrežne aktivnosti meri količino vseh prenešenih podatkov. Končne vrednosti so podane v tabeli 4.3.

Kot omenjeno v prejšnjem delu je bila metoda upodabljanja na GPE sposobna prikazati lepšo vizualizacijo in je za to potrebovala prenesti več podatkov. Prenašale so se bolj podrobne slike s podatki o višini in lahko smo prenašali tudi bolj podrobne slike ortofoto sloja. Večino so nanesle slike ortofoto sloja, s katerimi smo se trudili čimbolj podrobno prikazati površje terena. Skušali smo optimizirati količino prenešenih podatkov z natančnim omejeva-

Tabela 4.3: Meritve količine prenešenih podatkov. Vrednosti so navedene v megabajtih (MB). Manjše je boljše.

	GPE	CPE
Začetni pogled	6,1	**3,9**
Po treh premikih	45,4	**19,32**

Tabela 4.4: Meritve časa potrebnega za prenos podatkov. Vrednosti so navedene v sekundah (s). Manjše je boljše.

	GPE	CPE
Začetni pogled	**11,98**	12,95
Po treh premikih	**51,36**	68,58

njem ločljivosti prenešenih slik, ampak smo za doseganje želene kakovosti prikazovanja še vedno potrebovali prenesti slike z dovolj veliko ločljivostjo, sploh na območju, ki se prikaže ob povečavi.

Naslednji parameter, ki je povezan z omrežno aktivnostjo, je čas potreben za prenos in upodobitev podatkov. Spet smo meritev izvajali večkrat v podobnih pogojih, pri čemer velja omeniti, da je meritev pri zagonu aplikacije veliko bolj točna in relevantna kot meritev po treh premikih, saj je bil skupni čas nalaganja podatkov po premiku v veliki meri odvisen od uporabnika, ki se je pri testiranju moral ročno premikati s pogledom po terenu in tako sprožiti ponovno nalaganje. Podatki so podani v tabeli 4.4.

Pri meritvah časov nalaganja podatkov in vzpostavljanja sistema preseneča, da se je na prvi pogled občutno hitreje naložil sistem, uporabljen pri metodi upodabljanja na GPE, kljub temu, da je bilo potrebno prenesti več podatkov. Krajši čas nalaganja sistema gre pripisati manjši količini zahtevnih operacij izvedenih na CPE pred in med nalaganjem potrebnih po-

Tabela 4.5: Meritve količine porabljenega delovnega spomina. Vrednosti so navedene v megabajtih (MB). Manjše je boljše.

	GPE	CPE
Začetni pogled	**117,3**	269,2
Po treh premikih	**130,7**	1199,8

datkov, kar je omogočalo hitrejše izvajanje klicev za pridobivanje podatkov. Potrebno je omeniti, da meritve časa nalaganja podatkov po treh premikih niso relevantne, ker so bile močno odvisne od človeškega faktorja, saj smo morali premike izvajati ročno in smo tako lahko sami povzročili hitrejše ali počasnejše nalaganje novih podatkov.

4.3 Analiza porabe pomnilnika

Na koncu preglejmo še meritve parametra, ki lahko igra ključno vlogo pri uporabi aplikacije na šibkejših napravah. Meritev porabe pomnilnika nam pove veliko o učinkovitosti algoritma za upodabljanje, saj se da na tem mestu najti veliko prihrankov na enostaven način. Več o tem v komentarju rezultatov v zadnjem poglavju, same podatke pa predstavljamo v tabeli 4.5.

Poglavje 5

Sklepne ugotovitve

V sklepnem delu izpeljemo zaključke iz analize, ki smo jo opravili v prejšnjem poglavju. Ugotovitve nam pokažejo nekaj dejstev, ki jih bomo orisali in argumentirali v prvem oddelku tega poglavja, kasneje pa bomo predstavili še nekaj možnosti za izboljšave in nadaljnje delo.

5.1 Glavni rezultati

Kot smo začrtali v namenu magistrske naloge, nam je uspelo implementirati upodabljanja natančnega modela terena v spletni aplikaciji z ustreznimi programi na odjemalcu in strežniku. Uspešno smo rešili upodabljanje terena na več nivojih podrobnosti s pomočjo prirejene metode iz članka [18] in tako dosegli učinkovito prikazovanje in interaktivno uporabniško izkušnjo. Učinkovitost upodabljanja smo preverili s primerjanjem implementirane metode in obstoječih rešitev iz znanih knjižnic.

Zaključimo lahko, da se pri upodabljanju terena iz regularne mreže višinskih podatkov bolje obnese upodabljanje na GPE s prilagojenimi senčilniki in generiranjem prilagojenih slik s podatki o višini ter prilagojenimi slikami sloja, ki ga želimo prikazati na izrisani geometriji.

5.2 Predlogi izboljšav

Omenili bomo nekaj možnosti za izboljšave, ki smo jih med implementacijo opazili ampak jih nismo uspeli razviti. Med najbolj očitnimi možnostmi izboljšav je dodelava uporabniškega vmesnika, saj se temu v magistrskem delu nismo bolj podrobno posvečali. Nadalje bi lahko sistem predelali v platformo za izrisovanje dodatnih slojev na upodobljeni geometriji. Tako bi lahko poleg ortofoto sloja, ki je sedaj prikazan, prikazali še druge rastlinske, geološke in geografske sloje. Še bolj napredna razširitev bi lahko podpirala izris drugih 3D objektov nad prikazano geometrijo, vendar bi za bolj natančen prikaz morali še bolj podrobno prikazati teren in s tem žrtvovati učinkovitost upodabljanja, s katero zdaj deluje sistem.

Pomembna možnost izboljšave obstaja tudi v načinu obdelave zahtevkov za slike s podatki o višini. Strežnik, ki obdeluje te zahtevke, bi lahko z implementacijo standarda HTTP/2 [43] že po prvem zahtevku pravilno postregel z vsemi ostalimi slikami, brez da bi čakal na vsak zahtevek posebej. Za to bi bilo potrebno konkretno predelati celoten sistem nalaganja slik, poleg vseh predelav programa na odjemalcu pa bi moral tudi program na strežniku poznati logiko nalaganja slik s podatki pri upodabljanju na odjemalcu. Tako bi prišli do močno sklopljenega sistema, kar samo po sebi ni izboljšava, ampak mogoče bi napredek v hitrosti upodabljanja odtehtal slabšo arhitekturo aplikacije.

Dodatek A

Povezave

- URL izdelane spletne aplikacije: `http://212.235.189.233:8888/`
- URL repozitorija z izvorno kodo aplikacije: `https://github.com/rokrupnik/slo3d`

Literatura

[1] M. Pauly, M. Gross, L. P. Kobbelt, Efficient simplification of point-sampled surfaces, in: Proceedings of the conference on Visualization'02, IEEE Computer Society, 2002, pp. 163–170.

[2] D. P. Luebke, A developer's survey of polygonal simplification algorithms, IEEE Computer Graphics and Applications 21 (3) (2001) 24–35.

[3] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, O. Ruiz, Interactive visualization of volumetric data with webgl in real-time, in: Proceedings of the 16th International Conference on 3D Web Technology, ACM, 2011, pp. 137–146.

[4] N. Rego, D. Koes, 3dmol. js: molecular visualization with webgl, Bioinformatics 31 (8) (2014) 1322–1324.

[5] C. Marion, J. Jomier, Real-time collaborative scientific webgl visualization with websocket, in: Proceedings of the 17th international conference on 3D web technology, ACM, 2012, pp. 47–50.

[6] R. Pajarola, Large scale terrain visualization using the restricted quadtree triangulation, in: Visualization'98. Proceedings, IEEE, 1998, pp. 19–26.

[7] J. Dollner, K. Baumann, K. Hinrichs, Texturing techniques for terrain visualization, in: Visualization 2000. Proceedings, IEEE, 2000, pp. 227–234.

[8] C. Zhu, E. C. Tan, K. Chan, 3d terrain visualization for web gis, Map Asia (2003) 13–15.

[9] R. Olanda, M. Pérez, J. M. Orduña, S. Rueda, Terrain data compression using wavelet-tiled pyramids for online 3d terrain visualization, International Journal of Geographical Information Science 28 (2) (2014) 407–425.

[10] S. Rusinkiewicz, M. Levoy, Qsplat: A multiresolution point rendering system for large meshes, in: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343–352.

[11] A. Krooks, J. Kahkonen, L. Lehto, P. Latvala, M. Karjalainen, E. Honkavaara, Webgl visualisation of 3d environmental models based on finnish open geospatial data sets, The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 40 (3) (2014) 163.

[12] B. Štular, Ž. Kokalj, K. Oštir, L. Nuninger, Visualization of lidar-derived relief models for detection of archaeological features, Journal of archaeological science 39 (11) (2012) 3354–3360.

[13] T. Kjeldsen, P. T. Mikkelsen, J. Mosegaard, Making digital elevation modelsaccessible, comprehensible, and engaging through real-time visualization, Geoforum Perspektiv 14 (25).

[14] B. Resch, R. Wohlfahrt, C. Wosniok, Web-based 4d visualization of marine geo-data using webgl, Cartography and Geographic Information Science 41 (3) (2014) 235–247.

[15] H. Qiu, L. Chen, G. Qiu, H. Yang, An effective visualization method for large-scale terrain dataset, WSEAS Trans. Inf. Sci. Appl 10 (5) (2013) 149–158.

[16] H. Hoppe, Smooth view-dependent level-of-detail control and its application to terrain rendering, in: Visualization'98. Proceedings, IEEE, 1998, pp. 35–42.

[17] J. P. Suarez, A. Trujillo, J. M. Santana, M. de la Calle, D. Gomez-Deck, An efficient terrain level of detail implementation for mobile devices and performance study, Computers, Environment and Urban Systems 52 (2015) 21–33.

[18] F. Strugar, Continuous distance-dependent level of detail for rendering heightmaps, Journal of graphics, GPU, and game tools 14 (4) (2009) 57–74.

[19] P. B. Lundquist, Light detection and ranging, uS Patent App. 13/916,081 (Jun. 12 2013).

[20] S. E. Reutebuch, H.-E. Andersen, R. J. McGaughey, Light detection and ranging (lidar): an emerging tool for multiple resource inventory, Journal of Forestry 103 (6) (2005) 286–292.

[21] A. Neuenschwander, M. Crawford, C. Weed, R. Gutierrez, Extraction of digital elevation models for airborne laser terrain mapping data, in: Geoscience and Remote Sensing Symposium, 2000. Proceedings. IGARSS 2000. IEEE 2000 International, Vol. 5, IEEE, 2000, pp. 2305–2307.

[22] F. J. Aguilar, J. Mills, et al., Accuracy assessment of lidar-derived digital elevation models, The Photogrammetric Record 23 (122) (2008) 148–169.

[23] Z. H. Bowen, R. G. Waltermire, Evaluation of light detection and ranging (lidar) for measuring river corridor topography, JAWRA Journal of the American Water Resources Association 38 (1) (2002) 33–41.

[24] S. E. Reutebuch, R. J. McGaughey, H.-E. Andersen, W. W. Carson, Accuracy of a high-resolution lidar terrain model under a conifer forest canopy, Canadian journal of remote sensing 29 (5) (2003) 527–535.

[25] Projekt 'lasersko skeniranje in aerofotografiranje 2011', http://www.gu.gov.si/fileadmin/gu.gov.si/pageuploads/novice/Teksti_novic/LIDAR_opis.pdf, accessed: 2017-07-28.

[26] Izvedba laserskega skeniranja slovenije, blok 35 – tehnično poročilo o izdelavi izdelkov, http://gis.arso.gov.si/related/lidar_porocila/b_35_izdelava_izdelkov.pdf, accessed: 2017-08-28.

[27] D. Mongus, M. T. Cekada, B. Zalik, The analysis of an automatic method for digital terrain model generation from lidar data on slovenian test cases, Geodetski Vestnik 57 (2).

[28] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo, J. Blat, 3d graphics on the web: A survey, Computers & Graphics 41 (2014) 43–61.

[29] H. Hoppe, Progressive meshes, in: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM, 1996, pp. 99–108.

[30] A. Maglo, C. Courbet, P. Alliez, C. Hudelot, Progressive compression of manifold polygon meshes, Computers & Graphics 36 (5) (2012) 349–359.

[31] M. Limper, S. Wagner, C. Stein, Y. Jung, A. Stork, Fast delivery of 3d web content: a case study, in: Proceedings of the 18th International Conference on 3D Web Technology, ACM, 2013, pp. 11–17.

[32] G. Lavoué, L. Chevalier, F. Dupont, Streaming compressed 3d data on the web using javascript and webgl, in: Proceedings of the 18th international conference on 3D web technology, ACM, 2013, pp. 19–27.

[33] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, in: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.

[34] J. Jankowski, S. Ressler, K. Sons, Y. Jung, J. Behr, P. Slusallek, Declarative integration of interactive 3d graphics into the world-wide web: Principles, current approaches, and research agenda, in: Proceedings of the 18th International Conference on 3D Web Technology, ACM, 2013, pp. 39–45.

[35] D. Jackson, J. Gilbert, Webgl specification, `https://www.khronos.org/registry/webgl/specs/latest/1.0/`, accessed: 2017-08-04 (2014).

[36] Agencija rs za okolje, portal lidar, `http://gis.arso.gov.si/evode/profile.aspx?id=atlas_voda_Lidar@Arso`, accessed: 2017-07-28.

[37] Proj4js track and wiki, `http://trac.osgeo.org/proj4js/`, accessed: 2017-08-17.

[38] Spatial reference, `http://spatialreference.org/`, accessed: 2017-08-12.

[39] M. C. Doggett, Displacement Mapping and Volume Rendering Graphics Hardware, Universal-Publishers, 2002.

[40] W. Donnelly, Per-pixel displacement mapping with distance functions, GPU gems 2 (22) (2005) 3.

[41] Lod terrain, `https://github.com/felixpalmer/lod-terrain`, accessed: 2017-07-30.

[42] stats.js, `https://github.com/mrdoob/stats.js/`, accessed: 2017-08-11.

[43] M. Belshe, M. Thomson, R. Peon, Hypertext transfer protocol version 2 (http/2).